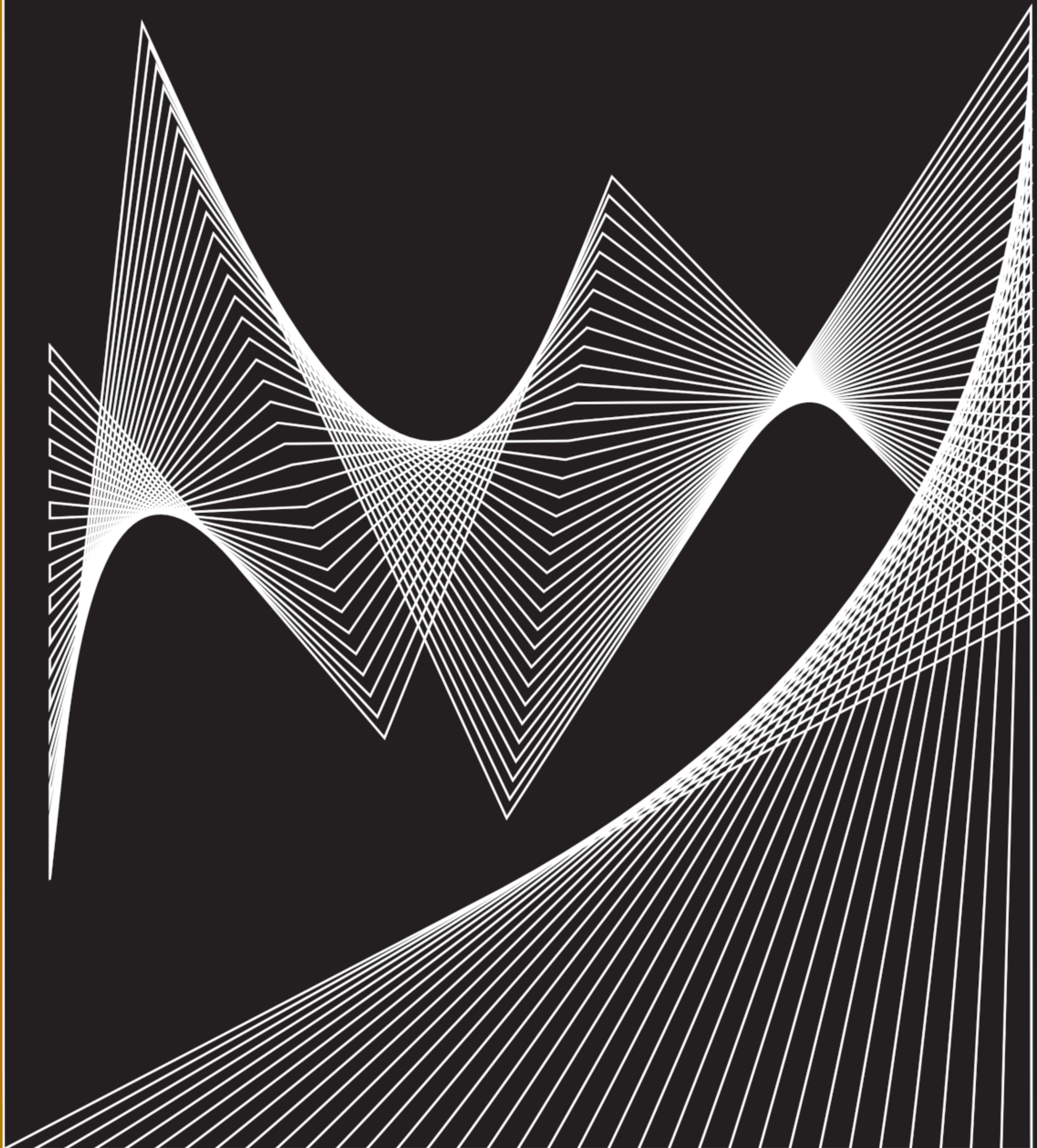
MSOR connections

Articles, case studies and opinion pieces relating to innovative learning, teaching, assessment and support in Mathematics, Statistics and Operational Research in HE.

Volume 18 No. 2

Special issue on programming in the undergraduate mathematics curriculum



Contents

For information about how to submit an article, notifications of new issues and further information relating to *MSOR Connections*, please visit https://journals.gre.ac.uk/index.php/msor.

This journal is published with the support of the **sigma** network and the Greenwich Maths Centre.

Editorial

Peter Rowlett, Department of Engineering and Mathematics, Sheffield Hallam University, Sheffield, UK. Email: p.rowlett@shu.ac.uk.
Matthew M. Jones, Design Engineering and Mathematics Department, Middlesex University, London, UK. Email: m.m.jones@mdx.ac.uk. (Guest editor)

This special issue of MSOR Connections is based on contributions to a workshop, ‘Programming in the undergraduate mathematics curriculum’, which took place at Middlesex University on 27th June 2019, and other submissions received on the same theme.

The Bond Review (*The Era of Mathematics: An Independent Review of Knowledge Exchange in the Mathematical Sciences*, 2018) recommended that “all mathematics students should acquire a working knowledge of at least one programming language.” In response to this, when the Institute of Mathematics and its Applications issued a call for proposals for its *Higher Education Teaching and Learning Series* workshops in 2019 this included a suggested theme “Developing undergraduate programming and coding skills in the mathematical sciences”. Matt Jones (Middlesex University) and Peter Rowlett (Sheffield Hallam University) submitted a proposal on this theme, which was funded by the IMA and Middlesex University and resulted in the workshop. We therefore gratefully acknowledge the role of the Bond Review and the Institute of Mathematics and its Applications in the origins of this special issue.

The first three papers report talks presented at the workshop; the remaining papers were received following an open call for submissions on the theme.

The issue opens with an article from Lynch detailing the change in the landscape of mathematics degrees and the impact of that on curriculum design around programming.

Rowlett’s article follows with a discussion of the art of programming as it relates to mathematical thinking and, in particular, the difference between coding and programming in the context of a second-year module. Next, Jones and Megeney continue the theme of developing higher programming skills with a case study detailing a group assessment on a second-year module where students are introduced to design patterns and version control allowing for a more significant piece of software.

Gwynllyw, Henderson, Van lent, and Guillot follow, presenting a case study also detailing how assessment can be designed to enable students to produce a more significant piece of software, this time in a third-year numerical analysis module.

Lee and Button follow with an update on the changes to the Further Mathematics A-level curriculum, in particular the Further Pure with Technology unit that introduces programming.

Three papers complete the issue presenting various automated assessment methods. Both Bostelmann and Morley detail the use of unit testing to assess student work, one in Java and the other in Python. Graham presents an article on assessing programming using the Numbas system.

Finally, to round off the issue, Elwes and Sturman present a set of barriers and how they were overcome in embedding computational mathematics in an undergraduate programme.

We write this in the middle of the worldwide Covid-19 pandemic. Over the last few months thousands have contracted and died of this virus including members of our community. Entire countries have been placed under various degrees of lockdown. And, as governments slowly ease restrictions, we face an uncertain future both in terms of our collective physical and mental health, and in terms of the economic impact.

In this context, we are especially grateful to the authors and anonymous reviewers of articles for this issue, who have worked in a relatively timely manner despite global circumstances.

We are grateful, too, for the work of Tony Mann and Alun Owen on editing this issue, particularly around the careful handling of articles from Rowlett and Jones and Megeney.

During the pandemic lockdown, there have been a number of online events related to university mathematics teaching, learning, assessment and support. We would like to draw your attention to session recordings and resources from:

- Teaching And Learning Mathematics Online (TALMO) [IMA, LMS and RSS] http://talmo.uk/
- **sigma** Online Support Workshop [**sigma** Network] http://www.sigma-network.ac.uk/sigma-online-support-workshop-29th-may-2020/
- E-Assessment in Mathematical Sciences (EAMS) [Newcastle University] https://eams.ncl.ac.uk/

Over the course of 2019, the editors and editorial board of MSOR Connections have completed a piece of work looking at the membership and makeup of the editorial board. This is the first issue published since the announcement of the enlarged and much more international editorial board, so we would take this opportunity to welcome Shazia Ahmed, Noel-Ann Bradshaw, Cosette Crisan, Anthony Cronin, Francis Duah, Jonathan Gillard, Michael Liebendörfer, Birgit Loch, Ciarán Mac an Bhaird, Eabhnat Ni Fhloinn, Josef Rebenda and Frode Rønning to the editorial board.

You can contribute to the work of *MSOR Connections* in providing a forum for sharing and discussion of ideas around teaching, learning, assessment and support by writing case studies about your practice, accounts of your research and discussing your opinions, and by acting as a peer reviewer for articles.

To submit an article or register as a reviewer, just go to https://journals.gre.ac.uk/index.php/msor. When you register as a reviewer, it is very helpful if you write something in the ‘reviewing interests’ box, so that when we are selecting reviewers for a paper we can know what sorts of articles you feel comfortable reviewing.

We hope you enjoy reading this issue.

CASE STUDY

Programming in the Mathematics Curriculum at Manchester Metropolitan University

Stephen Lynch, Department of Computing and Mathematics, Manchester Metropolitan University, Manchester, UK. Email: s.lynch@mmu.ac.uk. https://orcid.org/0000-0002-4183-5122.

Abstract

An increasing number of schools are teaching programming to their pupils and there is also an increase in programming in Higher Education with recent reports recommending this approach. At Manchester Metropolitan University (MMU) we wanted to attract and retain mathematics students and prepare them for careers upon graduation. By integrating Mathematics/Statistics/Operational Research packages across the curriculum and by solving real world problems we have managed to make the course highly desirable and loved by our students. In this case study, we show how it is possible to integrate programming and mathematical/computational modelling across the curriculum.

Keywords: Coding, computational modelling, real-world problems.

1. Introduction

In the 1990s, the mathematics degrees at MMU were technology based due to its historical polytechnic status. As technology became more and more advanced the staff found that they needed this technology to solve real-world problems. At that time, staff were using Maple™, MATLAB® and Mathematica® for mathematics, Minitab® for statistics and, for operational research, staff had developed their own software. There was not much programming involved in the course and students were being shown how technology could help solve problems in isolated pockets of the curriculum. After Curriculum 2000 was introduced into the UK, numbers on mathematics degrees across the country, including at MMU, dipped alarmingly and that is when we at MMU decided that a radical new approach was needed to draw students onto our mathematics degrees. It was also at this time that MMU decided to save money and pay for only one software licence for the faculty. Engineering has always been a much larger department than mathematics at MMU, and so it was decided that the university would adopt MATLAB as its sole programming language for all departments in the Faculty of Science and Engineering. In our new curriculum in 2000, we adopted MATLAB in mathematics and established a new first year unit entitled “Linear Algebra and Programming Skills.” The other mathematics units were also adapted to incorporate MATLAB, the statistics units continued with Minitab, and as operational research staff retired, some MATLAB and open-source software was adopted. As the curriculum continues to develop, we have found that programming is playing an increasingly important role within our courses.

Initially, there was some resistance from some of the students with regards to programming and the majority of students chose units without much coding, however, as staff became more experienced at teaching programming, there was a slow migration over to those units and now programming is popular on the mathematics degrees. We were delighted to read the government reports published in 2018 which vindicated our approach to teaching.

In February 2018, the Government Office for Science and the Council for Science Technology published the Blackett Review (Peplow, 2018), a report highlighting the importance of simulation utilizing computational and mathematical modelling of complex systems in both the public and private sectors. A few months later, in April 2018, an independent review of knowledge exchange in the mathematical sciences was published entitled "*The Era of Mathematics - Review Findings on Knowledge Exchange in the Mathematical Sciences"* (Bond, 2018). Amongst the findings and recommendations of the Bond review was the recommendation that – "*All mathematics students should acquire a working knowledge of at least one programming language*." Both of these reports highlight the importance of computational modelling and programming in the modern world. With mathematics packages such as Maple, MATLAB, Mathematica and Python; statistics packages such as Minitab, R, SAS, SPSS; and operational research packages such as CPLEX, FICO Xpress and Gurobi, mathematicians have the tools necessary to respond to these needs. With regards to computational modelling, the reader may be interested in the following simulation packages MapleSim^{TM}, Simulink$^{®}$ and Wolfram SystemModeler$^{®}$ developed by Maplesoft^{TM}, MathWorks$^{®}$ and Wolfram$^{®}$, respectively. Python does have a framework for modelling and simulating dynamical systems called SimuPy, but this has only recently been developed. For big data, machine learning and deep learning, the reader is directed to PyTorch and TensorFlow, and finally, for operational research simulation, Simul8 currently appears to be the most popular tool.

At Visit Days and Open Days at MMU we are finding an increasing number of students have experience at programming. Scratch is used in many primary and secondary schools around the UK. As of October 2019, the community statistics on the Scratch web pages show over 45,000,000 projects shared with more than 47,000,000 users registered (Scratch MIT, 2019). The Raspberry Pi is used in many secondary schools around the UK. It promotes both Python and Scratch as the main programming languages and there is also support for other languages. The official magazine for the Raspberry Pi is the MagPi magazine and there are regular Raspberry Jam events held in the UK and across the world. Amongst the many projects available to school children are: Replace your desktop PC with a Raspberry Pi; Print with Your Raspberry Pi; Set Up a Retro Gaming Machine; Build a Minecraft Game Server; Control a Robot; Broadcast a Pirate FM Radio Station; Build a Raspberry Pi Web Server and Learn How to Code. See (Balon, 2019) for more examples of the Raspberry Pi in education.

Programming in mathematics in Higher Education is becoming increasingly important in this technological age. In 2017, Sangwin and O'Toole (2017) conducted an online survey, with significant follow up correspondence, to establish the number of mathematics courses within the UK which incorporate programming on their degree program. The survey was completed by 63% of those institutons which offer a BSc (Hons) Mathematics degree. It was found that 78% of respondents included programming in a compulsory module and 11% did not teach programming at all. There are some universities in the USA which integrate programming into the undergraduate mathematics courses, see for example, (Buteau et al., 2015), (Cline et al., 2019) and (Jones et al., 2019).

2. Curriculum Design

At MMU, we have incorporated programming and computational modelling into the curriculum for more than twenty years. Figure 2 shows a typical four-year MMU MMath options map for the curriculum. The boxes coloured blue are heavily mathematics package-based, and packages such as Maple, Mathematica, MATLAB and Python would all serve equally well. The boxes coloured green are statistics based and boxes coloured yellow are operational research based. Suitable packages are again indicated in the legends in the top right-hand corner of Figure 1.

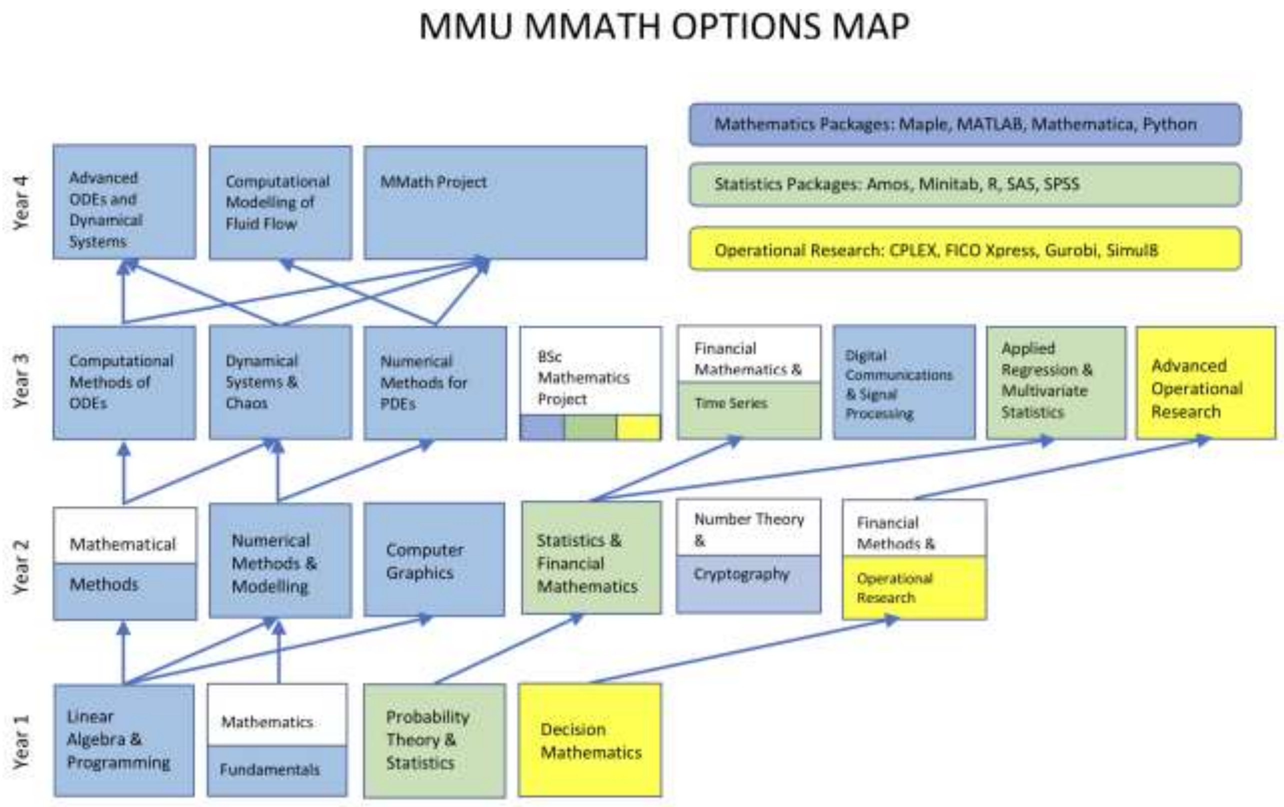

MMU MMATH OPTIONS MAP
Mathematics Packages: Maple, MATLAB, Mathematica, Python
Statistics Packages: Amos, Minitab, R, SAS, SPSS
Operational Research: CPLEX, FICO Xpress, Gurobi, Simul8
Year 4
Advanced ODEs and Dynamical Systems
Computational Modelling of Fluid Flow
MMath Project
Year 3
Computational Methods of ODEs
Dynamical Systems & Chaos
Numerical Methods for PDEs
BSc Mathematics Project
Financial Mathematics & Time Series
Digital Communications & Signal Processing
Applied Regression & Multivariate Statistics
Advanced Operational Research
Year 2
Mathematical Methods
Numerical Methods & Modelling
Computer Graphics
Statistics & Financial Mathematics
Number Theory & Cryptography
Financial Methods & Operational Research
Year 1
Linear Algebra & Programming
Mathematics Fundamentals
Probability Theory & Statistics
Decision Mathematics

Figure 1. A typical curriculum map for Mathematics modules at MMU indicating that programming and computational modelling underpin the whole curriculum. Each module (block) is worth 30 credits. The emphasis is on programming for Mathematics and not programming for Computer Science.

The key unit in the curriculum map in Figure 1 is the Linear Algebra and Programming unit in Year 1. The students are first shown how to use MATLAB/Python as a graphing calculator and in every Linear Algebra class, MATLAB/Python functions are used to solve simple problems and plot figures. Within a few weeks, students are introduced to three programming constructs: (i) defining functions; (ii) for and while loops; and (iii) if, then, elif, else statements. The students are assessed using two in-class tests and one summer examination. One of the in-class tests set in a laboratory is based on simple programming ensuring that the students understand the three basic programming constructs (i) – (iii) above. Towards the end of the unit, programming is used in Linear Algebra to demonstrate linear transformations, in particular, rotation and translation of objects in two-dimensional space. This provides a precursor to the Year 2 unit entitled Computer Graphics – and students are given an insight into what is studied there. MATLAB/Python is also used as a graphing calculator in the other Year 1 units where specialist software in statistics and operational research is also introduced for computational modelling.

In Year 2, MATLAB/Python is used in all six units at various levels and the students are shown how programming and computational modelling can be used throughout the mathematics curriculum.

It is in Years 3 and 4 that the students start to see the real benefits of programming and computational modelling. The units Computational Methods of ODEs, Dynamical Systems and Chaos, Numerical Methods for PDEs and Digital Communications and Sound Processing are heavily MATLAB/Python-based. The coursework for these Year 3 units require some extensive programming and interpretation of solutions to real-world problems and the examinations take place in a computer laboratory with access to MATLAB/Python. Statistics and operational research packages are used for computational modelling in the units Financial Mathematics and Time Series, Applied Regression and Multivariate Statistics and Advanced Operational Research but there is far less programming in these units. This allows students a route through the curriculum if they are averse to programming. There is an extensive list of mathematics projects offered in Year 3 – some are heavily programming/computational modelling based whilst others avoid programming and modelling all together. In Year 4, the units Advanced ODEs and Dynamical Systems, Computational Modelling of Fluid Flow and the MMath project are all heavily programming/computational modelling based and the students are expected to reproduce results from recently published journal papers. In the first two units, the examinations again take place in a computer laboratory. In the next section, we will show exemplars of coursework and examination questions used at MMU.

3. Learning and Teaching and Assessment

A typical 30 credit unit will consist of one/two hours of lectures followed by two/one hours in a computer laboratory. Lectures can have between 30-120 students present and the computer laboratories have a typical capacity for 30 students - this means that some laboratory sessions have to be repeated. Staff have found that teaching in a computer laboratory is the best medium to get to know the students more intimately – better even than one-to-one tutorials. The students happily talk and help one another and there is generally a very relaxed atmosphere enabling closer interactions. Students seem very happy with this method of teaching, since the inception of the Student Union Teaching Awards in 2011, mathematics has been shortlisted on six separate occasions winning the award in 2012 and 2018. In 2015, Mathematics at MMU was visited by a Teaching Fellow form University College London (UCL) and an IEEE conference paper was published in 2016 comparing the incorporation of programming into curricula at both MMU and UCL (Nyamapfene, 2016). Some exemplars of examination and coursework questions involving programming and computational modelling will now be listed for illustration.

As Linear Algebra and Programming Skills are so vital to the curriculum, we start with an examination question from that Year-1 unit.

Examination Question 1:

(a) A geometric progression has first term *a* and common ratio $r < 1$, and the sum of the first *n* terms is given by the expression:

$$S_n = \frac{a(1-r^n)}{1-r}.$$

Write a MATLAB/Python program which can input the first term, a, and compute and output the sum of the values of r between $r = 0.1$ and $r = 0.7$ in steps of 0.2, and values of n between $n = 5$ and $n = 20$, in steps of 5. The results should appear in three columns for r, n and S_n, with suitable column headings. The value of r should be printed to one decimal place and the sum, S_n, to five decimal places. The program should use two nested **for loops**. Save the file as **exam1a.m** or **exam1a.py**, and run the program with $a = 4$, and output the results to the file **exam1a.txt**.

[Program 8 marks, Result 2 marks]

(b) Modify your part (a) program such that values of a and r are inputted from the keyboard and use a **while loop** to compute the sum, S_n, for a range on n values, starting at $n = 3$, and increasing in steps of 3. The loop should continue while the absolute difference between the part (a) sum and the sum to infinity, $S_\infty = \frac{a}{1-r}$, is greater than 0.5×10^{-6}. The table now only needs to show the values of n and S_n, again shown to five decimal places. Save the program as **exam1b.m** or **exam1b.py**, and run the program with $a = 10$ and $r = 0.5$, and output the results to the file **exam1b.txt**.

[Program 8 marks, Result 2 marks]

The next question is taken from a Year-2 examination for Operational Research and Financial Mathematics.

Examination Question 2:

The local government operational research team are developing a Markov Chain model to describe the way that the city residential population is changing. It is predicted that on average 5% of suburbanites will want to move into the centre each year and 3% of centre dwellers will want to move to the suburbs. Currently, 30% of the population live in the centre and 70% live in the suburbs.

(a) (i) Draw a Markov Chain model for the problem and write down the single stage transition matrix for it. (ii) Write a MATLAB/Python program that determines the number of years required for the proportions of centre dwellers and suburbanites to become stable and state the stable proportions. Work to four decimal places and save your file as **exam2a.m** or **exam2a.py**.

[10]

(b) A flat screen TV manufacturer offers a complete replacement warranty if a TV fails within 2 years. Based on compiled data the company has noted that 4% of the TVs fail during the first year. Whereas, 1% of TVs that survive the first year will fail during the second year. The warranty does not cover replacement TVs. (i) Draw a network representation of the problem. Hint you should have four states. (ii) Derive the fundamental matrix *N* and matrix *B*. (iii) From the matrix *B,* give an estimate of the number of TVs that will require replacing.

[10]

The next question is a coursework question from the Year-3 unit Applied Regression and Multivariate Analysis.

Coursework Question 1:

Suppose that a random variable *Y* satisfies $Y \sim Bin(3, p)$. In this case,

$$E[Y] = 3p, Var[Y] = 3p(1-p).$$

In contrast to the above, a quasibinomial model can be constructed that satisfies

$$E[Y] = 3p, Var[Y] = 3p(1-p)\phi.$$

(a) For what values of ϕ is this distribution: (i) Over-dispersed? (ii) Under-dispersed?

(b) Show that $R = 1 + Y$, satisfies

$$E[R] = 1 + 3p, Var[Y] = 3p(1-p)\phi,$$

where the mean and variance of Y can be derived from part (a). Table 1 relates the Research Excellence Framework ratings of submitted journal papers against the journal ranking using the Association of Business Schools journal list.

(c) Enter this data into R using the labels ABS and REF, and include a screenshot of your R workspace once you have done this.

Table 1: Research Excellence Framework Data

REF Rating	ABS 4*	ABS 3*	ABS 2*	ABS 1*	Unrated
4*	94	80	4	2	3
3*	95	296	29	1	6
2*	47	150	54	9	37
1*	3	28	10	6	21

(d) Using the command **sample(1:975, 1)**, delete one of the datapoints at random from each of the series ABS and REF. Include the details in your coursework submission.

(e) Using a logistic quasibinomial model test for a relationship between y and the ABS journal rating, determine the effect of including a "-1"-term in the formula for the explanatory variables on the right-hand side of the equation.

(f) Using the model output in part (e), copy and complete the following table:

ABS Rating	E[R]	Var[R]
4*		
3*		
2*		
1*		
Unrated		

[TOTAL=25 Marks]

The final examplar in this section presents a coursework question for the Year-4 unit, Advanced Ordinary Differential Equations and Dynamical Systems.

Coursework Question 2:

Consider the following nonlinear Leslie population model, where the decay rates decay exponentially with population size and the population is observed once per year:

$$X^{(k+1)} = \begin{pmatrix} b_1 e^{-rN} & b_2 e^{-rN} & b_3 e^{-rN} \\ c_1 & 0 & 0 \\ 0 & c_2 & 0 \end{pmatrix} X^{(k)},$$

where $X = (x, y, z)^T \in R^3$, is a vector of the k'th age class populations.

(a) Given that $b_1 = 5, b_2 = 50, c_1 = 0.8, c_2 = 0.6, r = 0.1, N = x + y + z,$ determine the fixed points of period one and their stability.

(b) Using the same parameters as in (a), given that, for the scaled population $x^{(0)} = 5, y^{(0)} = 15,$ and $z^{(0)} = 5$: (i) compute the number of females in each age class after 100 years; (ii) plot a graph showing how the populations evolve and give a physical interpretation.

(c) Given that $b_1 = 50, b_2 = 50, c_1 = 0.8, c_2 = 0.6, r = 0.1, N = x + y + z,$ (i) use MATLAB/Python to produce an animation in 3-dimensional space which shows how the phase portrait changes as the parameter $b_2$increases from $b_2 = 50$ to $b_2 = 80$. Initially take, $x^{(0)} = 5, y^{(0)} = 15,$ and $z^{(0)} = 5$; (ii) take still shots of the animation to submit your coursework and describe what happens to the age class populations physically.

[TOTAL=25 Marks]

On reflection, teaching in lectures, tutorials and computer laboratories provides a great deal of variety for both our students and staff and we believe that this variety is a major factor in what makes the mathematics degree loved by our students. The assessments can be made more challenging and can be applied to real-world problems, which the students appreciate. Finally, the students and parents who come to us for Open Days and Visit Days tell us that the highlight of the day is the interactive Python sessions in our laboratories.

4. Conclusion and Future Enterprises

The course philosophy for mathematics at MMU is to provide an applications-based approach using computational modelling and programming to solve real-world problems. The curriculum map displayed in Figure 1 and discussed in Section 2 demonstrates how this is possible. Section 3 presents exemplars of coursework and examination questions for a number of units from Years 1 to 4. Interested readers will find my Python book (Lynch, 2018) useful for material for Dynamical Systems units in Years 3 and 4. The book also includes a chapter listing 20 coursework exemplars as well as three example examination papers. From 2020, we will be offering an annual summer term workshop on Python programming for other departments in the Faculty of Science and Engineering at MMU. Working with Professor Louise Walker at the University of Manchester, we will also be offering annual summer workshops on "*Python for A-Level Mathematics and Beyond,*" suitable for both teachers and sixth form students. The goal is to get more students to choose mathematics at degree level.

5. Acknowledgements

This paper is based on a talk given at the ‘Programming in the Undergraduate Mathematics Curriculum’ workshop, Middlesex University, 27th June 2019.

6. References

Balon, B. and Simic, M., 2019. Using Raspberry Pi computers in education. *42nd International Convention on Information and Communication Technology, Electronics and Microelectronics*, pp.671-676. https://doi.org/10.23919/MIPRO.2019.8756967.

Bond, P. ed., 2018. *The Era of Mathematics – Review Findings on Knowledge Exchange in the Mathematical Sciences*. Engineering and Physical Sciences Research Council and the Knowledge Transfer Network. Available at: https://admin.ktn-uk.co.uk/app/uploads/2018/04/KE-booklet-for-web.pdf [Accessed 25 March 2020].

Buteau, C., Muller, E. and Ralph, B., 2015. Integration of programming in the undergraduate Mathematics program at Brock University. *Online Proceedings of the Maths + Coding Symposium*. London (Canada). Available at: http://researchideas.ca/coding/docs/ButeauMullerRalph-Coding+MathProceedings-FINAL.pdf [Accessed 25 March 2020].

Cline, K., Fasteen, J., Francis, A., Sullivan, E. and Wendt, T., 2019. Integrating programming across the undergraduate mathematics curriculum. *PRIMUS: Problems, Resources, and Issues in Mathematics Undergraduate Studies*, 30(7), pp.735-749. https://doi.org/10.1080/10511970.2019.1616637.

Jones, L.B. and Hopkins, B.J., 2019. Teaching a course in mathematical programming. *PRIMUS: Problems, Resources, and Issues in Mathematics Undergraduate Studies*. Available at https://doi.org/10.1080/10511970.2019.1619207.

Lynch, S., 2018. *Dynamical Systems with Applications using Python*. New York: Springer International Publishing.

Nyamapfene, A. and Lynch, S., 2016. Systematic integration of MATLAB into undergraduate mathematics teaching: Preliminary lessons from two UK institutions. *IEEE Educon 2016*, pp.1145-1148. https://doi.org/10.1109/EDUCON.2016.7474699.

Peplow, M. ed., 2018. *Computational Modelling: Technological Futures*. Government Office for Science and Council for Science and Technology. Available at https://www.gov.uk/government/publications/computational-modelling-blackett-review [Accessed 25 March 2020].

Sangwin, C.J. and O’Toole, C., 2017. Computer programming in the UK mathematics curriculum. *International Journal of Mathematical Education in Science and Technology*, 48(8), pp.1133-1152. https://doi.org/10.1080/0020739X.2017.1315186.

Scratch.mit.edu, 2019. Community statistics at a glance. Available at: https://scratch.mit.edu/statistics/ [Accessed 25 March 2020].

CASE STUDY

Programming as a mathematical activity

Peter Rowlett, Department of Engineering and Mathematics, Sheffield Hallam University, Sheffield, UK. Email: p.rowlett@shu.ac.uk. https://orcid.org/0000-0003-1917-7458.

Abstract

Programming in undergraduate mathematics is an opportunity to develop various mathematical skills. This paper outlines some topics covered in a second year, optional module ‘Programming with Mathematical Applications’ that develop mathematical thinking and involve mathematical activities, showing that practical programming can be taught to mathematicians as a mathematical skill.

Keywords: programming, mathematical thinking, applications.

1. Introduction

In a mathematics degree, choice of taught content should support the development of the skills needed by graduate mathematicians. Programming can play a considerable role here: directly, by involving mathematical computation and being related to many graduate jobs including in areas such as data science; and indirectly, being closely related to mathematical skills such as clear thinking, problem-solving and precise communication. This article discusses ways in which a module on programming has been used to develop mathematical skills. First, a description is given of the teaching context. Following this, mathematical teaching and assessment activities are discussed as involving mathematical thinking and mathematical applications.

2. A module ‘Programming with Mathematical Applications’

Programming is taught to undergraduate mathematics students via an optional second-year, 20-credit module, ‘Programming with Mathematical Applications’. This follows on from a little programming experience as part of a core first-year module, plus considerable variety of formal and informal prior knowledge. In two years of operation, the module has attracted between 20-30 students, who work on laptops around tables in an informal classroom environment. Teaching is mostly via written notes with code examples, motivated by diverse prior knowledge and the fact that students will learn a lot more from running their own code than watching me do so. The self-paced nature is commented on by students as a positive. For students already familiar with the content, a series of challenge problems are offered, as well as the freedom to explore programming interests beyond the taught curriculum.

The module comprises taught content on programming fundamentals and mathematical functionality, graphics, GUIs and data manipulation, presentation of mathematical information on the web, and database queries. The choice of topics is based somewhat around their popularity observed through roles taken by our graduates. Following the directly taught content, students take the lead via individual projects completed partly in class time with staff support via in-class discussion and two scheduled progress meetings.

Assessment is via staged independence. A first piece of coursework asks students to write a series of functions to perform specific tasks, for example “Write a function that takes as its input a number x and returns the value of $\frac{2x}{x-1}$”. A second coursework gives a detailed program specification and asks students to write a program to meet this functionality. The task is specified at a higher level than the first coursework, for example “Generate 10 HTML files containing certain pseudorandomised mathematical information". Finally, an individual project assignment gives a flexible brief or area of interest and allows the student greater freedom in demonstrating their programming abilities. Students were asked to submit a program (language not specified) along with a HTML page which contained: an explanation of the mathematical statistical or technical content; a description of how the program works; and, a critical review of the work.

3. Programming as mathematical thinking

3.1. Programming vs. coding

There are lots of program constructs that are common to many programming languages, such as if statements, loops and functions. I define *programming* as the process of using program constructs to express an algorithm in a way that a computer can interpret and act on. *Coding*, by contrast, is the business of writing a program in the syntax of a particular language.

Programming, requiring clear, systematic thinking and unambiguous communication, is a skill closely linked to mathematics and the practice of the professional mathematician. In teaching mathematics students, it seems appropriate to focus principally on programming. In order to do so, though, it is necessary to write code in a particular language. I tell students that the fact we are coding in a particular language is secondary; the important thing is that they are learning how to program. This also has the advantage that learning to understand program constructs and the process of combining these to implement an algorithm is a skill that can be readily transferred to other programming languages.

I tell my students syntax errors are essentially inconsequential, in order to emphasise programming skills over coding. Rote learning of specific syntax is not a worthwhile goal of programming teaching. People who know several languages might sometimes use the syntax from one in another, particularly if they are using a language they haven't used for a while. This is a minor issue, easily overcome. I tell students they should not feel they are failing as programmers if they have to search for help with syntax. I say that learning how a for loop works and what it can be used for is important, but remembering the precise syntax to make one happen in a particular language is secondary. I also say that if they are regularly working in a particular language, they will find they start to memorise syntax for that language much more easily. If they stop using a language for a while, they will become rusty on syntax, but the idea is that their understanding of the programming fundamentals will persist.

3.2. Teaching directed to programming that isn't coding

I deliberately open the first class in my programming module with a piece of teaching that does not use computers. We are in a standard teaching room with laptops available at discretion of the lecturer, so telling students they are not getting laptops out is a stunt, designed to focus attention on programming skills beyond writing code. (You might feel it is similarly a stunt to have the single reference in an article about programming be from 1896, or indeed to wait until two thirds of the way through the article before naming the language in which students are coding.)

I give students a paper worksheet on propositional logic that opens with this statement:

> Computers do what you tell them to do, not what you really *meant* them to do, so when programming it helps to think strictly logically. This worksheet is designed to refresh some relevant concepts and encourage you to think clearly and carefully.

The worksheet covers propositions and predicates, NOT, AND, OR, XOR and truth tables, and contains some exercises designed to emphasise clear communication, examples of which are given below.

Students are asked to identify propositions – statements that have a truth value – such as “Two plus two equals four” and “The moon is made of green cheese”, from that list that includes statements that are not propositions, such as “Would you like a cup of tea?”

Students are asked to think about the difference between OR and XOR (exclusive OR) in some natural language sentences, such as “Did you see Claire or Alex earlier?” and “Is the door open or closed?” An example exercise in thinking and communicating clearly is given in figure 1(a). In this, social convention dictates that the first ‘or’ is an XOR, while the second is an OR. There is room for interesting discussion here, because while the waiter may intend the first as an XOR, it isn’t strictly the case because if the customer answered “both” to the first question, they would presumably be served both (by the principle ‘the customer is always right’).

Another truth table activity relates to implication. Say p is “$5 = 7$” and q is “you get a first class degree”. Students are asked to consider $p \Rightarrow q$ using a truth table. Then they are asked to resolve some sorites by Lewis Carroll, which are a series of statements which can be linked to form a single chain of predicates. An example is given in figure 1(b) (taken from Carroll, 1896, p. 112). This is designed to support clear thinking and lead into the right kind of thinking for programming logic using if statements.

(a) Consider the following conversation.

```
WAITER: Would you like tea or coffee?

CUSTOMER: Tea, please.

WAITER: Would you like milk or sugar?

CUSTOMER: Both, please.
```

Which type of ‘or’, OR or XOR, is being used by the waiter in each of the two questions?

(b) Write each of the following three statements as an implication.

(i) All babies are illogical.

(ii) Nobody is despised who can manage a crocodile.

(iii) Illogical persons are despised.

Arrange these implications into a chain so that you reach a consistent conclusion.

Figure 1. Sample exercises targeted at precise communication and clear thinking.

Later in the first class, after the students have spent some time getting Python up and running and writing a ‘Hello, World!’ program, another worksheet is based around algorithms. I tell students algorithms are a sequence of precise instructions. In an electives choice session, where students are presented with details about each optional module in order to choose which to study, I show a picture of some cakes and tell them these were produced by my son and me by following an algorithm. In class, I ask students to write algorithms for every day and mathematical activities, such as “putting on a t-shirt”, “making a cup of tea” and “differentiating the product of two functions”. I then ask them to show their algorithm to someone else, with the instruction:

> When you are shown someone else’s, try to be as pedantic as possible and wilfully misinterpret the instructions. The point is to think about how to really tightly specify a sequence of actions.

This is a fun activity, and a little silly. Some students are reluctant, but I believe benefit from being pushed to participate. A reluctant student might simply say “put your arms in the arm holes and your head through the head hole”. A response might be “but now the t-shirt is inside out or back-to-front!” A particularly witty student I observed answered this kind of loosely-specified algorithm with the question “isn’t it uncomfortable to be wearing a t-shirt that has a coat hanger inside?”

4. Programming using mathematical applications

As well as embedding precise and mathematical thinking in activities, there are plenty of opportunities to make use of mathematical applications in programming exercises.

Some example mathematical activities used in examples and exercises while learning and practising basic programming:

- calculating whether a given year is a leap year (if statements);
- generating terms of the Fibonacci sequence (for loop);
- prime factorisation (while loop);
- computing factorials (recursion);
- asking the user to input a number with some property (user input);
- import data from a CSV file into a spreadsheet (file access);
- generating charts from data using a GUI interface (graphics).

Further mathematical and statistical functionality is included via Python modules, and programming skills can be practised while using these, for example:

- math: mathematical functions, e.g. log, sin;
- cmath: mathematical functions for complex numbers;
- numpy: support for matrices and related mathematical functions;
- scipy: scientific computing, e.g. ODE solvers, linear algebra, etc.;
- matplotlib: 2D and 3D plotting;
- pandas: data analysis tools;
- sympy: symbolic computation.

It is possible to include mathematical applications when testing programming skills in slightly more complicated tasks, for example:

- simple operations: convert a complex number from Cartesian to polar coordinates.
- statistical methods: hypothesis testing.
- numerical methods: find the eigenvalues of a given matrix.
- brute-force number theory: testing for certain properties of a number, e.g. whether it is abundant.
- simulation: simulate flipping a coin 1000 times and report how many heads occurred and what was the longest chain of heads in a row.

Project-based learning, involving more in-depth, open-ended tasks, is an opportunity to include more advanced mathematical activities while assessing programming skills. Students are given a choice of programming language for this task. Most choose either Python or VBA, which are taught in the module, with a small number choosing alternative software such as Matlab. Some example project areas are given below.

- Exploring computer functionality, e.g. investigation of how numbers are represented in computers and the resultant errors that occur; methods for generating pseudo-random numbers; manipulation and calculation of dates.
- Numerical or computational methods, e.g. showcasing linear algebra or ODE solver methods, statistical methods or data analysis tools.
- Simulation/modelling, e.g. cellular automata, scheduling a sports league or tournament; iterative prisoner's dilemma; programming a simple combinatorial game.
- Mathematical investigation, e.g. fractals; representations of 3D objects; machine learning, e.g. via artificial neural networks or generic algorithms.
- Historical investigation, e.g. implementing Ada Lovelace's program for computing Bernoulli numbers; investigating methods for approximating π.

5. Discussion

There are many opportunities not explored here. Partly, this is because our degree is in general quite computational. For example, students make use of Matlab in mathematical modelling modules and of statistical and data analysis software in statistics teaching. Even so, there is much opportunity to develop the skills of a graduate mathematician through programming, as I hope this article has conveyed.

6. Acknowledgements

This paper is based on a talk given at the ‘Programming in the Undergraduate Mathematics Curriculum’ workshop, Middlesex University, 27th June 2019.

7. References

Carroll, L., 1896. *Symbolic Logic: Part I Elementary*. London: Macmillan.

CASE STUDY

Programming in Groups: developing industry-facing software development skills in the undergraduate mathematics curriculum

Matthew M. Jones, Department of Design Engineering and Mathematics, Middlesex University, London, UK. Email: m.m.jones@mdx.ac.uk.
Alison Megeney, Department of Design Engineering and Mathematics, Middlesex University, London, UK. Email: a.megeney@mdx.ac.uk.

Abstract

Programming is increasingly becoming an expected graduate skill for mathematics students. We argue in this article that programming should be given the same priority as any other graduate skill. Given the practical and philosophical constraints placed on undergraduate mathematics curricula, however, we acknowledge the difficulty in introducing, in a meaningful way, many of the core ideas of programming. We therefore present a case study of a second year course on an undergraduate mathematics programme that introduces Object Oriented Programming and aspects of software design, as well as key practical skill such as version control. We will argue that group assessment in this context is a more natural setting for students to be working and reflects more closely the experience of programming in industry; furthermore, it serves as a convenient platform to introduce students to aspects of software design and practical programming considerations. We will present an example of the type of assessment that can be used and how Version Control Systems like Git can be used to give students a more realistic experience of programming with the advantage of allowing tutors and other group members to track student work.

Keywords: Programming, Group assessment, Employability, Graduate Skills

1. Programming as a Graduate Skill

Historically, computing has been embraced by mathematicians as a tool for studying and solving problems in mathematics. The introduction of the NAG Libraries for FORTRAN in 1970 and TeX in 1979 serve as very early examples of its contributions. In different ways both of these had a significant impact on mathematics. However, they also highlight a common attitude of mathematicians to programming. According to Sangwin and O'Toole (2017) programming, as currently taught in undergraduate mathematics curricula across UK HEIs, largely reflects this natural order, often being introduced and taught in mathematics courses as a tool for solving specific problems: numerical solutions to ODEs/PDEs, numerical analysis, mathematical and statistical modelling and many other areas. This is likely the reason why the authors' findings suggest that languages such as MATLAB or R are amongst the more popular languages taught. The authors highlight a number of gaps, however, in the current offering by mathematics departments, not least the fact that programming paradigms such as Object Oriented Programming or Functional Programming may not be introduced to students in any meaningful way (p. 1145):

> *It is therefore somewhat surprising that [programming paradigms] are not currently taught and since they are at best optional, the vast majority of undergraduate students will never encounter these programming paradigms as part of their undergraduate education.*

In contrast the most popular languages for computing degrees (see Murphy *et al.*, 2017) are Java, C (and its successors C++ and C#), and Python; R is not being taught at all and MATLAB is taught in fewer than 3% of computing courses. This difference is explained by the fact that computing degrees have a clearer career progression, however it is also likely due to the relationship many professional mathematicians have with programming: it is a tool for solving specific problems or simplifying calculations. This is corroborated in Murphy *et al.* (2017) where the authors asked respondents *why* they chose their particular language. The most prevalent response, independent of the language, was its relevance in industry. Since the second most popular career choice for mathematics graduates is IT, according to Prospects (2019), and programming is increasingly becoming an important skill, we argue that it should be considered a *graduate skill* and that as mathematicians we should be mindful of this in curriculum design, even to the point of taking the lead from computing degrees. However, we also appreciate that mathematics degrees are, of course, not computing degrees and there are a number of hurdles to introducing graduate skills in mathematics curricula. Indeed, as Waldock (2011, p. 5) says,

> *There are significant barriers involved when seeking to modify Mathematics programmes to encourage the development of graduate skills. One is fundamentally philosophical, as some will wish to retain the pure, theoretical nature of their courses. Another is the practical difficulty of finding space for graduate skill development in a crowded curriculum.*

The view of students entering degree programmes in the UK has changed significantly in the last 20 years. The days when a university degree was seen as the sole route to career success have gone. In the most recent Global Learner Survey (Pearson, 2019), only 17% of UK respondents agreed with the statement that a college degree is essential to achieving a successful and prosperous career. This demonstrates a significant shift from previous studies. For example, a YouGov poll in 2012 found that 81% of respondents thought going to university was essential for them to pursue their career (Adediran, 2015). Additionally, in the Global Learner Survey, 66% of UK respondents believed a degree or certificate from a vocational college or trade school is more likely to result in a good job with career prospects than a university degree.

These changes in student attitude come at a time when the STEM skills shortage is highly publicised and a source of concern. UK government policy has in the last 10-15 years attempted to close this gap, and the extent to which universities should be responsible for addressing the shortage has been controversial in areas such as mathematics. However, with the publication of the so-called Augur report in May 2019 (Department for Education, 2019), there is a clear move to a situation where degree value is measured by graduate prospects rather than on its own merit. As a result, it is likely that graduate skills will become ever more important and will need to be transparent in the curricula of mathematics degrees in the future.

In this climate it is, therefore, becoming necessary for subjects like mathematics to reaffirm their position as career-facing subjects and, we would suggest, challenge the complacency that mathematics is, by some measure, top-of-the-pile in terms of its employability status. It is with this in mind that we have reconsidered how we teach programming on undergraduate mathematics degrees at Middlesex University, aiming to include specific, industry standard skills training that students can highlight to potential employers. And we have done this in a way that minimises the encroachment into the standard curriculum.

2. Context

The course we discuss in this case study is a second year undergraduate course on the BSc Mathematics programme. Students learn either R or Python in their first year and are introduced to

Java in their second-year. As is recommended in Sangwin and O'Toole (2017) this design means programming is taught throughout the first two years of the students' degree rather than in isolated courses, and remains optional in their third year. The course in question is a skills-based course, *Problem Solving Methods*, that introduces students to a wide range of techniques in applied mathematics as well as techniques to develop mathematical problem solving skills in pure mathematics (see Jones and Megeney, 2018). Workshops are inquiry-led, sometimes employing the Moore method (see Parker, 2005), to encourage students to develop their problem-solving skills and confidence. The content of the module ranges from areas of applied mathematics including optimisation, mathematical modelling, numerical methods and analysis, to areas of pure mathematics including number theory and real analysis. Students work weekly on different problems, developing strategies to solve abstract and unfamiliar problems, building a set of robust, internalised tools for enquiry. Programming is used as one such tool for examining problems and conjecturing solutions, and students are encouraged to see it as one of many avenues of progress. The structure of the workshops is heavily influenced by Pólya (1957), although expanded to include, as tools for examining problems, the use of software or programming. Whereas when Pólya wrote his work on solving problems he wrote about examining examples to get a better understanding of a problem, we encourage students to do the same using computers. The use of programming thus becomes one of the many integral tools available to students to study problems.

Students arrive in their second year with a good grounding in basic procedural programming and have developed some appreciation and experience of algorithm design. Introduction to a new language is therefore a matter of learning a new syntax (although further specific differences must also be mastered such as might be expected when learning a compiled, statically typed language).

Although the course content is taught in an informal workshop setting, many of the initial programming laboratories are taught more traditionally. Topics are introduced by the tutor and students work in pairs using the driver/navigator model, as described in Hannay *et al.* (2009) and Brown and Wilson (2018). There is an emphasis on teaching students many of the formal concepts from computer science that are necessary to implement object oriented design principles. We do not aim to teach aspects of functional programming; although Java does incorporate this paradigm in some sense, the course team does not believe it is in the interest of the students to confuse object oriented programming and functional programming. The taxonomy outlined in Figure 1, influenced by Selby (2015) and our own experience, is used as reference; it models the cognitive journey and, especially, our aspirations for where they will reach. The Aesthetics alluded to in the figure are not taught explicitly – instead students see aspects of them in the problems they solve and the assessment. The assessment of the course consists of individual coursework and group coursework; it is the latter that we wish to discuss here.

Students become accustomed to working in teams and presenting their work in class. This helps alleviate some of the issues common in group work as discussed, for example, in MacBean *et al.* (2004).

3. Structure of the group assessment

The philosophy of object-oriented programming lends itself naturally to group work, compartmentalisation of code allows group members to work independently of one another whilst still being part of a team. Indeed, aspects of high-level design such as design patterns, abstraction and inheritance are given a heightened importance – students must *design* the structure of the programme *before* they start coding in order to maintain compatibility.

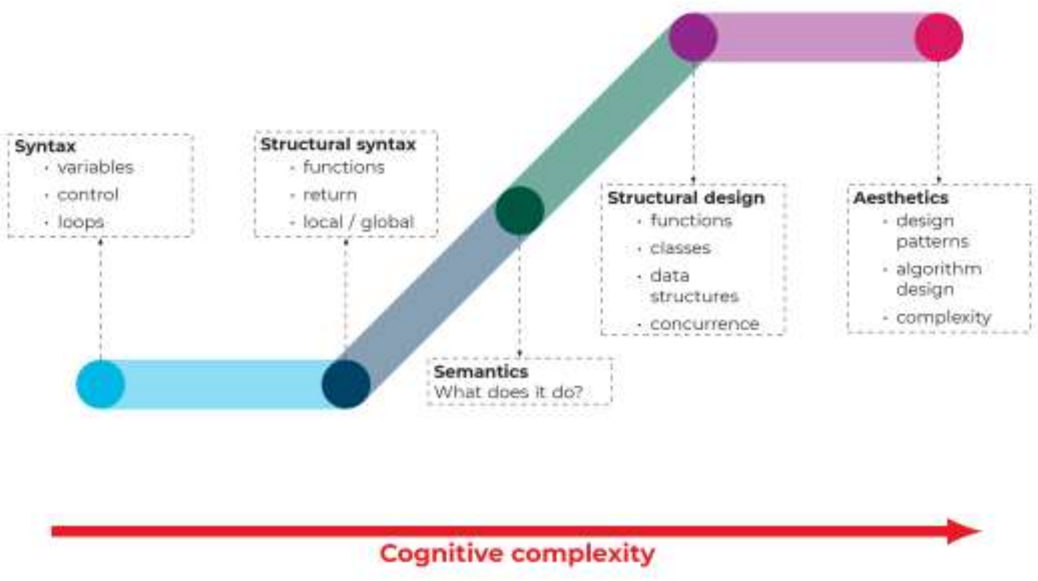


Figure 1: Programming taxonomy

In software design, design patterns are pre-packaged solutions to common problems, a classic reference to these is the so-called Gang-of-Four (Gamma *et al.*, 1995). In our opinion one of the most accessible design pattern for teaching is the Factory Design pattern (and, to a lesser extent, the Abstract Factory Design). We focus on this in our design of suitable assessment since it serves as a useful platform to further enquiry into design patterns. However it should also be noted that this is only our personal preference and other design patterns might also lend themselves naturally to group work.

The coding for the group assessment must be of the following form:

1. Decomposable into smaller problems
2. Each smaller problem should be solvable independently of the others
3. There should be an interface that ensures compatibility of code
4. The solution should lend itself to the Factory Design pattern (Figure 2).

Groups are arranged with a lead who will be responsible for the interface and acts as a client for the software – i.e. queries, runs and presents output. Other members of the group are responsible for solving the smaller parts of the problem.

As an example we might have groups write software that solves numerically an ordinary differential equations. Groups would normally be expected to use different numerical techniques such as Huen's method, or various other levels of precision of Runge-Kutta to find solutions. Individual group members can then take responsibility for each of the techniques used and the lead takes responsibility for the overall design.

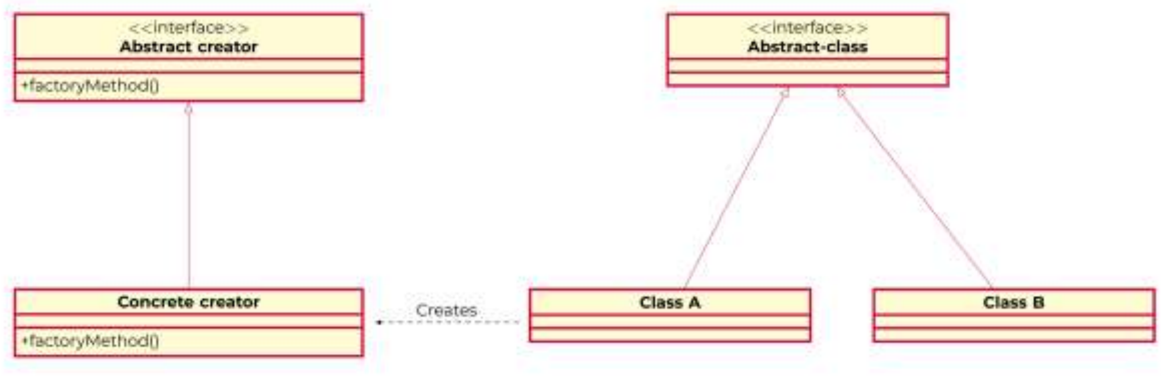


Figure 2: UML diagram for the factory design pattern

The advantage of structuring assessment in this form is that, with appropriate feedback, students will naturally discover that something approximating a factory design pattern must be used. Tutors then introduce students to Design Patterns or at least the Factory Design Pattern. Note at this stage students will have already encountered many design patterns in the course of their learning of object oriented programming, although may not have recognised them as such. For example the Iterator Design is built in to Java, and students will have seen the Adapter pattern when implementing data structures.

An important aspect of designing assessment that is decomposable like this is that students need to address the issue of version control – how does one know one is working on the current version of the code, or how does one avoid clashing with other work done. Version Control Systems are numerous, but the most common is Git. The university uses a local Git repository provided by GitLab. We do not advise students to use their own GitHub accounts to complete group assessment since elsewhere we promote GitHub as a convenient place for students to document a portfolio of work, so only final versions of software are made available on GitHub. Students are trained in the use of Git to maintain software, and the log from individual Git forks are used to ensure comparability of work effort in the final submission, thereby mitigating against the problem of 'coasting'.

4. Reflection and Concluding Remarks

Our approach to group assessment in programming discussed in this article is still in its initial stages with only two cohorts having been assessed this way. In a future article we expect to be able to evaluate the effectiveness using longitudinal data. However we are not yet at this stage.

Initially our main concern was that mathematics students would not feel comfortable learning technology like Git, especially those that were not aiming to go on to careers in software development. However, we have found that students have reacted positively to the opportunity of learning it. The following student's response summarises views on the usage of Git:

> *Being introduced to Git within the [undergraduate] degree is also helpful as the student can set up their own GitHub profile, load up their coursework and use this as a portfolio which is amazing for employability.*

In our experience students look favourably on opportunities to develop outward-facing exhibits of their work for employers and so even those not thinking of careers at this point see the advantage of using a tool like this.

We were also concerned that students would not be able to make the link between the structure of the assessment and the factory design pattern at all. Instead, in all cases, groups naturally designed their code with some of the notions of these patterns embedded. This helped improve confidence in their coding significantly and when, during formative feedback sessions, students were introduced to the formal factory design pattern it was evident that they made a significant connection to what can be an abstract idea. In some cases students went on to research more about design patterns and algorithm design.

Group work can be fraught with problems such as coasting, and a perceived increase in plagiarism. Indeed, there is growing scepticism amongst undergraduate students that it is worth the effort. Whereas plagiarism can be mitigated to some extent by assessment design, the coasting effect is certainly still an issue for some students using the approach described in this article. We prefer a proactive approach to dealing with these problems, intervening when needed. Communicating the use of Git to measure mutual effort has been useful, but we have not yet taken the approach of weighting group members' marks based on this. Given the approach described in this article is still evolving, we may well need to take a more authoritarian approach to this if necessary in the future.

In conclusion, considered as a pilot, our approach to introducing more advanced programming techniques in a group setting has been successful. In our experience students are well-suited to independently discover practical aspects of programming. It should be noted however that we have not yet encountered a situation where students have not independently discovered the ideas we have intended them to, and this will need to be considered in future.

5. Acknowledgments

This paper was presented at the "Programming in the Undergraduate Mathematics Curriculum" workshop held at Middlesex University on 27th June 2019. We would like to thank the sponsors of the event: IMA and Middlesex University.

6. References

Adediran, M., 2015. *Students value university education over costs*. Available at: https://yougov.co.uk/topics/politics/articles-reports/2015/05/01/students-value-university-education-over-costs [Accessed 30th January 2020].

Brown, N.C.C. and Wilson, G., 2018. *Ten quick tips for teaching programming*. PLoS Computational Biology, 14(4). https://doi.org/10.1371/journal.pcbi.1006023.

Department for Education, 2019. *Independent panel report to the Review of Post-18 Education and Funding*. Available at: https://assets.publishing.service.gov.uk [Accessed 30th January 2020].

Gamma, E., Helm, R., Johnson, R. and Vlissides, J., 1995. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley.

Hannay, J.E., Dybå, T., Arisholm, E., Sjøberg, D.I.K., 2009. The effectiveness of pair programming: a meta-analysis. *Information and Software Technology*, 51(7), pp. 1110-1122. https://doi.org/10.1016/j.infsof.2009.02.001.

Jones, M. and Megeney, A., 2018. Problem solving methods in undergraduate mathematics. In: *CETL-MSOR Conference 2018 Evidencing Excellence, 05-06 Sept 2018, University of Glasgow, Scotland*.

MacBean, J., Graham, T. and Sangwin, C., 2004. Group work in mathematics: a survey of students' experiences and attitudes. *Teaching Mathematics and its Applications*, 23(2), pp. 49-68. https://doi.org/10.1093/teamat/23.2.49.

Murphy, E., Crick, T. and Davenport, J.H., 2017. An Analysis of Introductory Programming Courses at UK Universities. *The Art, Science, and Engineering of Programming*, 1(2). https://doi.org/10.22152/programming-journal.org/2017/1/18.

Parker, J., 2005. *R. L. Moore: Mathematician and Teacher.* Mathematical Association of America.

Pearson, 2019. *The Global Learner Survey*. Available at: https://www.pearson.com/content/dam/global-store/global/resources/Pearson_Global_Learner_Survey_2019.pdf [Accessed 30th January 2020].

Pólya, G., 1957. *How to Solve It*. Princeton, NJ: Princeton University Press.

Prospects, 2019. *What do graduates do?* Available at: https://luminate.prospects.ac.uk/tag/reports [Accessed 30th January 2020].

Sangwin, C.J. and O'Toole, C., 2017. Computer programming in the UK mathematics curriculum. *International Journal of Mathematical Education in Science and Technology*, 48(8), pp.1133-1152. https://doi.org/10.1080/0020739X.2017.1315186.

Selby, C., 2015. Relationships: computational thinking, pedagogy of programming, and Bloom's Taxonomy. *The 10th Workshop in Primary and Secondary Computing Education, United Kingdom.* pp. 80-87. https://doi.org/10.1145/2818314.2818315.

Waldock, J., 2011. *Developing Graduate Skills in HE Mathematics Programmes - Case Studies of Successful Practice*. Birmingham: Maths, Stats and OR Network. Available at: http://www.mathcentre.ac.uk/resources/uploaded/gradskills.pdf [Accessed 30th January 2020].

CASE STUDY

Using Python in the Teaching of Numerical Analysis

D. Rhys Gwynllyw, Department of Engineering Design and Mathematics, University of the West of England, Bristol, UK. Email: rhys.gwynllyw@uwe.ac.uk.

Karen L. Henderson, Department of Engineering Design and Mathematics, University of the West of England, Bristol, UK. Email: karen.henderson@uwe.ac.uk.

Jan Van lent, Department of Engineering Design and Mathematics, University of the West of England, Bristol, UK. Email: jan.vanlent@uwe.ac.uk.

Elsa G. Guillot, Department of Engineering Design and Mathematics, University of the West of England, Bristol, UK. Email: elsa.guillot@uwe.ac.uk.

Abstract

In this Case Study we describe the rationale, methodology and results of teaching Python as part of a third year optional Numerical Analysis module taken by undergraduate BSc Mathematics students at the University of the West of England, Bristol. In particular we focus on how we have used programming mini-tasks to engage and prepare students for using Python to complete a more significant piece of coursework, taken later in the course. These mini-tasks are marked electronically using the Dewis e-assessment system which provides the students with immediate and tailored feedback on their Python code.

Keywords: Python, programming, e-assessment, numerical analysis.

1. Introduction

In a recent report, Bond (2018) recommended that computer programming becomes a core part of mathematics degrees. For many years, students on the BSc Mathematics course at the University of the West of England (UWE), Bristol used the Maple (2019) computing environment as a combination of a computer algebra system and a programming language. In the first two years of the three year course, Maple was used primarily as a symbolic engine and for visualisation of solutions. A short course on computer programming was included in one of the first year modules but, for the students' first two years of study, their use of Maple as a programming language was limited. However, in their third year, students taking the optional Numerical Analysis module were required to create and develop their own Maple programs.

In 2018 it was decided to introduce Python into the BSc Mathematics course. There were a number of reasons that led us to this decision. Firstly, although Maple can be used as a programming environment/language, it was felt that this was not its primary function. As such, we were not exposing students to a typical programming environment. Secondly, outside of Mathematics higher education, Maple is not a widely known package and hence a student's experience of Maple is not necessarily significant to potential employers. In addition, there is less information and support for Maple online (via GitHub for example, https://github.com/) than for other languages (RankRed, 2019). In contrast, Python has become one of the most popular programming languages (The Economist, 2018) and as with Maple, Python has capability for symbolic mathematics using the SymPy package (Meurer et al., 2017).

Having made the decision to introduce Python into the BSc Mathematics curriculum, we identified a suitable pilot course to be the optional Level 6 Numerical Analysis module. The module's design already accounted for some students' lack of confidence in computer programming by allocating part of the coursework to the completion of mini-tasks throughout the first semester. These mini-tasks provided the students with template code for them to alter to implement more challenging tasks. Keeping a similar structure for the mini-tasks meant that students would be able to get to grips with the basics of Python through a highly-scaffolded approach.

Our move to Python also meant that Dewis (2012) could be used to electronically assess the mini-tasks. Dewis is a fully algorithmic open-source web-based e-Assessment system which was designed and developed at UWE (Gwynllyw and Henderson, 2009). The e-assessment of computer programs had already been used at UWE for a number of years using the Dewis system in a project involving a collaboration of the Mathematics and Computer Science groups (Gwynllyw and Smith, 2018). In that project, the e-assessment of C-programs was performed with the Dewis system marking both the output and the structure of students' computer programs.

2. Methodology

2.1. Overview of the Numerical Analysis module

In this Case Study we considered the 30 credit optional module Numerical Analysis, which is available to final year students on the BSc (Hons) Mathematics course at UWE. The module covers the implementation and analysis of a number of numerical methods applied to a range of mathematical problems and is taught over the whole academic year. Each week, students attend a two hour lecture, and a one hour computing lab. Additionally, all students are timetabled to an optional one hour drop-in session for the module.

The first semester concentrates mainly on the numerical solution of initial value problems (IVPs) and covers the topics of

- Runge-Kutta (RK) and Linear Multistep methods (LMM);
- Error analysis – mostly local truncation error analysis (including adaptive time-stepping);
- Linear stability analysis.

Also in this semester students are introduced to the topic of the finite difference method applied to boundary value problems. The second semester concentrates on numerical solutions to partial differential equations.

2.2. Teaching Approach of Python

The implementation of the teaching of Python started in induction week, where all final year Mathematics students were invited to an 'Introduction to Python' course run by a team of academics from the mathematics group. Those final year students who had chosen the Numerical Analysis module were informed beforehand that participation at this day-long course was considered to be essential. This introduction to Python course was through the Spyder integrated development environment (IDE). This IDE was chosen due to its simplicity and shallow learning curve. Although this introductory course was not written specifically for the Numerical Analysis module, a part of the course content was motivated by the requirements of that module. The overall syllabus of this course was:

- creating and running simple scripts;
- basic commands in Python;

- the process of identifying and correcting bugs;
- variables and data structures (including lists);
- loops;
- logic and conditional statements;
- creating and using user-defined functions;
- importing libraries and files.

For most students this was their first exposure to Python. However all students would have been introduced to programming basics, albeit using Maple, in their first year at UWE.

In the Numerical Analysis module itself, the teaching and practice of Python was concentrated within the first semester. In the second semester, students were supplied with Python code to implement the methods with very little coding changes required on their part. Standard Python was used wherever possible to implement and analyse numerical methods without including additional libraries.

For implementing the actual numerical schemes (RK and LMM), as described in Section 2.1, students were supplied with three files, as shown in Table 1, that they were required to modify.

Table 1. Details of the three Python files provided to students.

File name	Description
main.py	The main program, which sets the numerical parameters, defines the initial value problem and implements the method.
method.py	Contains the function ‘calculate’ that defines the numerical method. This function receives the problem and numerical parameters and returns arrays containing the numerical results.
output.py	Contains functions that present the numerical results, both as files containing data values and as graphical output. The graphical output uses the ‘matplotlib’ library.

The Numerical Analysis module introduces students to different aspects of computer programming. For example, throughout the semester, students were exposed to the use of loops, both of fixed number of iterations and, for the case of variable time-stepping, conditional (while) loops. Linear stability analysis exercises required students to determine exponential growth/decay and hence, in producing graphical output, they had to consider the use of log scales. Students were required to use complex variables in their computation of linear stability threshold values (e.g. having expressed a coupled system of real-valued initial value problems as a pair of decoupled, possibly complex-valued, initial value problems). In order to implement the numerical methods, students needed to be proficient in the use of arrays (lists). This involved the case of static arrays (fixed known length typically for the case of fixed time steps) and dynamic arrays (typically for the case of variable time-step methods) and to recognise differences in their manipulation. Together with the use of arrays (lists), students are exposed to the use of list comprehension (constructing arrays in a concise manner in one expression incorporating loops and conditional statements). In the coverage of boundary value problems, students had to identify the representation of a matrix by a list of lists. In addition they were introduced to functions from the ‘numpy.linalg’ library for solving a linear system. We should note that the Numerical Analysis module was not intended to represent a fully comprehensive course on programming, but instead the important basic programming skills relevant for numerical analysis. New concepts were introduced when relevant to the material being covered.

2.3. Assessment overview

The assessment in this module is composed of an end-of-year exam (75% of the module mark) and a coursework (25% of the module mark). The coursework is partitioned into two parts as follows:

Part One: worth 20% of the coursework mark takes the form of four mini-programming tasks and has four staggered deadline dates throughout the first semester.

The main aims of these mini-tasks are to

- encourage student engagement with the programming throughout the first semester;
- prepare students for the more significant programming task in Part Two of the coursework.

Part Two: worth 80% of the coursework mark is a hybrid written report/programming coursework with a mid-February deadline. Most of this coursework involves a given mathematical problem (typically in the form of coupled IVPs) and a given numerical method. Students are required to construct the numerical method in a form suitable to the problem. They are required to perform both theoretical and empirical analysis of the results. This analysis may include stability and/or local truncation error (LTE) considerations and require the students to comment on unusual behaviours in their results.

2.4. e-Assessment of Python

Given the main purposes of Part One of the coursework (engagement and preparation) we felt it was essential that the marking and feedback for all four mini-tasks to be as fast and supportive as possible (Race, 2014). We had employed this partitioning of the coursework model for a number of years (using Maple). Previously a manual marking process was employed but the workload involved in processing these student submissions, resulted in difficulties in producing timely feedback. In the conversion from Maple to Python, it was decided to implement instantaneous electronic marking and feedback of the Part One mini-tasks in order to address the above problem. The previous deployment of Dewis to e-assess computer code in C (Gwynllyw and Smith, 2018) meant that the development time required for Dewis to e-assess Python was significantly reduced. In fact, most of the Dewis code for marking C programs is the same for Python. Necessary alterations to the code included the implementation of a new set of ‘banned keywords’ for Python, that is, keywords in the student’s submission that were not allowed for security reasons. With Python being an interpreted language, as opposed to C being a compiled language, there were other changes required to ensure that memory and CPU limiters were applied to any execution of the student’s code on the Dewis server. It should be stated here that any such execution is made in a sandbox environment on the Dewis server to protect the server from malicious attack.

To access the mini-tasks, students were directed to log-in to the University’s virtual learning environment (Blackboard) and access the appropriate Dewis assessment for the mini-task (using an automatically authenticated Learning Tools Interoperability link). Students were given three attempts at each of these mini-tasks. The Dewis system included an error-checker on the student’s submission so that, if a student’s code would not run on the Dewis server, then Dewis would report back the Python error report as part of the feedback. In such cases, we instructed Dewis not to decrement the number of submission attempts left for the student. Further details of each of the mini-tasks is included in the next section.

2.5. Particulars of the Python Mini-Tasks

All four tasks in Part One of the coursework relate to numerical methods applied to the solving of the first order initial value problem:

$$\frac{dy}{dt} = f(t, y), \qquad y(t_0) = y_0\,, \tag{1}$$

or a coupled form of (1). Currently these four mini-tasks are as follows:

Mini-task 1: This is the only task that does not require the student to submit Python code. The purpose of this task is simply to ensure that students know how to operate Spyder and understand the structure of the supplied code. The student is given access to Python files for implementing Euler's method to solve (1). The supplied Python files implement Euler for specific values of the problem parameters $f(t,y)$, t_0, y_0 and numerical parameters h and n. On attempting an e-assessment, the student is given different values of these parameters and they are asked to obtain the numerical approximation of $y(t)$ for a given t. Therefore the student is tasked with altering the 'main.py' file (see Table 1), to run their code in Spyder and to recognise which time-stage (i) corresponds to the required value of $y(t)$.

Mini-task 2: For this task students are required to alter the code supplied to them in mini-task 1 so that the code implements the Modified Euler method. Specifically, the student is required to submit to Dewis their modification of the 'method.py' file which contains the function

calculate (f, t_0, t_n, y_0, n). (2)

where the system parameters in (2) are as for the initial value problem (1) and f represents a Python function. The numerical parameter t_n is the final time value and n is the number of time steps. These two numerical parameters thus determine the size of the time-step, h.

With regards altering the supplied code, the students were instructed that they needed to only alter the interior body of the 'calcuate' function in (2), that is, they were required to leave the function's parameter listing and return types (the arrays $[t_i], [y_i]$) unaltered.

On submission of their 'method.py' Python code into Dewis, the system performs some security checks and then runs the student's code *four* times using the following instances of the derivative function $f(t,y)$:

$$(i)\;\; f(t,y) = 0, \quad (ii)\;\; f(t,y) = c_1, \quad (iii)\;\; f(t,y) = \cos(c_2 t), \quad (iv)\;\; f(t,y) = \cos(c_3 t) + \sin(c_4\, y),$$

where $c_1,\ c_2,\ c_3$ and c_4 are constants randomly generated by Dewis.

For each instance (i)-(iv) of running the student's code on the Dewis server, Dewis also runs the corresponding model solution code and compares the $[t_i], [y_i]$ output arrays from the two runs. If the generated arrays have the same values, then the student's code is deemed to have been successful in its running of the modified Euler method for that particular derivative function. However, the student only receives full marks if all four runs are deemed successful.

The purpose of the four distinct runs is to help provide tailored feedback for cases where the student code does not achieve full marks. In such cases, we want students to investigate for themselves the reason for any errors. However, the four runs give a mechanism for suggesting to the student the areas of their error. For example, if the student's code gives correct results for derivative functions (i)-(iii), but not for (iv), the feedback would suggest that the student investigate whether the y dependency in the f function has introduced the error. The student would be advised to check the values they used in the second parameter of any call made to the $f(t,y)$ function.

Mini-task 3: Students are required to alter the code supplied to them in mini-task 1 so that the code implements the Runge-Kutta 3/8 code for a *coupled system*. Specifically, the student is required to submit to Dewis their modification of the 'method.py' file to contain a function

$$\text{calculate }(f,\, g,\, t_0,\, t_n,\, y_0,\, u_0,\, n) \tag{3}$$

to numerically solve the coupled system

$$\frac{dy}{dt} = f(t,y,u); \quad y(t_0) = y_0;$$

$$\frac{du}{dt} = g(t,y,u); \quad u(t_0) = u_0.$$

The students are instructed that function (3) is required to return the arrays $[t_i], [y_i], [u_i]$.

The marking process in mini-task 3 is similar to that in mini-task 2 in that four different runs are performed with different characteristics of the f, g functions in order to facilitate the feedback in the event of any errors. For example, it is only in the fourth run that the case of a fully coupled system is considered, whilst the third run corresponds to the case of two *decoupled* problems, i.e., effectively $f = f(t,y)$ and $g = g(t,u)$.

Mini-task 4: Students are required to alter the code supplied to them in mini-task 1 so that the code implements an Adams-Bashforth-Moulton (ABM) method (3/2-step) together with Heun's 3rd order method as the 'start-up' method (two time-steps). In addition, students are required to implement this method in as efficient a manner as possible with respect to the number of calls made to the derivative function $f(t,y)$.

The marking process in mini-task 4 is an extension of that used in mini-task 2. The same function types are used but, in addition, for the case of errors occurring, Dewis investigates whether the student errors occurred in the Heun start-up method or in the subsequent ABM method.

In addition, each running of the student code is checked for efficiency. In running the student's code, Dewis monitors the number of times the derivative function is called and hence measures the efficiency of the student's code in this regard. The reason for doing this is that, with the ABM method being a linear multistep method, the number of calls to the derivative function can be reduced significantly by storing the most recent values of the derivative within variables (or an array) that is updated at each time-step. This is a desirable approach and one which the students were encouraged to take.

In terms of the marking of this mini-task, three of the five marks were awarded for correct results for the $[y_i]$ array and two of the marks were awarded for the efficiency of the implementation. If a student's scheme was close to optimal efficiency their submission would be awarded one of these two marks.

3. Results

With Part One of the coursework, students were strongly recommended to use the weekly drop-in session to discuss any issues they had with their submissions. The intent of this part of the coursework was to encourage participation and discussion. At any time during which these mini-tasks were open, students could access their Dewis feedback which included a link to retrieve their submitted code. This facilitated the feedback process with students using this feature to discuss their code submission with the academic on duty.

At the end of the suite of Part One mini-tasks, we investigated all the cases where students' code was in error, to determine whether the feedback they received was appropriate to their submission. This was found to be the case. In addition, we recognised further possible avenues for enhancing feedback which is to be implemented in future years. Further additions to the mini-tasks are planned based on our experience in marking Part Two of the coursework. For example, from marking the 2019/2020 coursework, it seems clear that students struggle to implement a LTE estimator based on Richardson's extrapolation. Hence we plan to include an additional mini-task for obtaining LTE estimators. Our experience of common student errors in the coding of LTE estimators will be used in the design of this task's marking and feedback mechanisms.

4. Discussion

Following the successful introduction of Python into the Numerical Analysis course, it was decided to teach Python, instead of Maple, to our first year Mathematics students in the 2019/20 academic year. Python was taught within an existing Calculus and Numerical Methods course and students attended a two hour computer practical class every week for the first semester. The aim of the course was to learn Python while performing mathematical investigations. Students were introduced to the SymPy, NumPy and Matplotlib libraries which provide extra commands for symbolic and numerical calculations and plotting. Towards the end of the course programming concepts were introduced such as functions, conditional statements and loops.

Python has been used for the two most recent runs of the Numerical Analysis module. For both runs, the final year students have had extensive exposure to Maple at previous levels, but neither have benefitted from our newly introduced Python teaching in the first year. Hence, it is encouraging to note that we have not detected any decrease in student performance nor understanding in programming when compared with previous years' module runs (when Maple was used). This is evidenced by average coursework marks and pass rates for this module being at similar levels to previous years. Student feedback from end of year module evaluations shows that the use of Python has been very positive; students recognise the importance of learning a programming language that is relevant to industry. Students have also stated that they appreciate the Dewis-generated feedback augmented by tutor support in the drop-in sessions.

Having introduced Python in the first year of the BSc Mathematics course the students will develop their skills further through its implementation in second year modules from the 2020/21 academic year. Further, the teaching of Python is embedded into our new problem based learning curriculum which will roll out from 2020/21 onwards; Dewis mini tasks will be used to support students learn Python in their first year, which will enable more challenging tasks to be set throughout the rest of their studies.

5. References

Bond, P. ed., 2018. *The Era of Mathematics – Review Findings on Knowledge Exchange in the Mathematical Sciences*. Engineering and Physical Sciences Research Council and the Knowledge Transfer Network. Available at: https://epsrc.ukri.org/newsevents/pubs/era-of-maths/ [Accessed 4 March 2020].

Dewis Development Team, 2012. Dewis welcome page. Available at: http://dewis.uwe.ac.uk [Accessed 3 March 2020].

Gwynllyw, R. and Henderson, K., 2009. DEWIS: a computer aided assessment system for mathematics and statistics. *CETL-MSOR 2008 Conference Proceedings*. pp. 38-44.

Gwynllyw, R. and Smith J., 2018. E-Assessment of Computer Programming. In *Proceedings of 12th International Symposium on Advances in Technology Education*.

Maple, 2019. Available at: https://maplesoft.com/products/Maple/ [Accessed 3 March 2020].

Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R.P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M.J., Terrel, A.R., Roučka, Š., Saboo, A., Fernando, I., Kulal, S., Cimrman, R. and Scopatz, A., 2017. SymPy: symbolic computing in Python. *PeerJ Computer Science* 3:e103. Available at: https://doi.org/10.7717/peerj-cs.103 [Accessed 10 March 2020].

Race, P., 2014. *Making Learning Happen*. Sage Publications.

RankRed, 2019. *Python Is Now The Second Most Popular Language On GitHub*. Available at: https://www.rankred.com/python-the-second-most-popular-language/ [Accessed 5 March 2020].

The Economist, 2018. *Python is becoming the world's most popular coding language*. Available at: https://www.economist.com/graphic-detail/2018/07/26/python-is-becoming-the-worlds-most-popular-coding-language [Accessed 4 March 2020].

SHORT UPDATE

Programming in the recently-updated pre-university maths curriculum – Further Pure Mathematics with Technology (FPT)

Stephen Lee, Mathematics in Education and Industry, Trowbridge, UK. Email: Stephen.lee@mei.org.uk.
Tom Button, Mathematics in Education and Industry, Trowbridge, UK. Email: Tom.button@mei.org.uk.

Keywords: A level, curriculum, programming, pure, technology.

1. Background

In the article ‘Moving with the times – a new A level Further Mathematics Unit: Further Pure Mathematics with Technology (FPT)’, Lee and Button (2013) wrote about the optional A level Further Mathematics unit that featured programming. Since then both GCSE and A level Mathematics/Further Mathematics have been revised (see Baldwin and Lee, 2017, and Glaister, 2017).

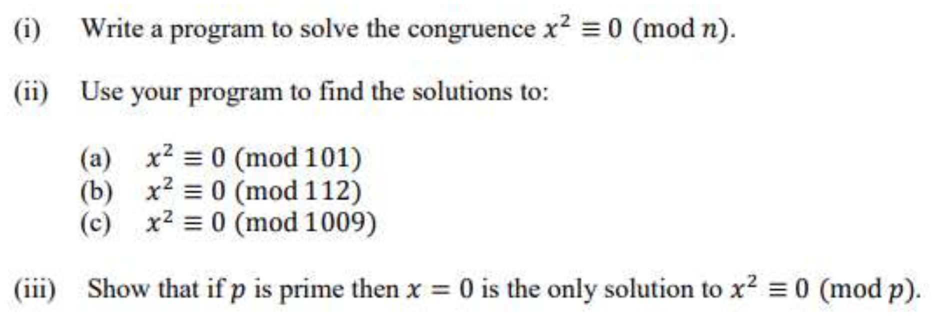
A level Mathematics and Further Mathematics changed for first teaching in academic year 2017/18 and thus, after completing these maths qualifications over two years, those students would have predominantly entered university in the current academic year 2019/20 (Note: a small number may have completed the new courses in one year and entered university in 2018/19). A level Mathematics is still the most popular of all subjects and in summer 2019 there were 91,895 entries in the UK; 14,527 studied A level Further Mathematics (Joint Council for Qualifications data). However, this was a decrease of 5.9% and 10.1% respectively from 2018.

The major change is that A levels are no longer modular courses; students are assessed by end of course examinations. The content of A level Mathematics is completely prescribed; however, the content of A level Further Mathematics still retains some element of choice over some of the content. The Further Pure with Technology unit has been updated and is still available as an optional component for Further Mathematics students and in this short update we will address its role and how it can support pre-university students in developing their mathematical programming skills.

2. Aim and philosophy of FPT

The aim and philosophy of FPT is to support students in developing their skills in using technology for pure mathematics. There are three topics studied: investigation of curves, differential equations and number theory. In investigation of curves students use a graph-plotter and Computer Algebra System (CAS); in differential equations they use a graph-plotter, CAS and a spreadsheet; in number theory they use a programming language (they are required to use Python in this course). The common approach in all of these topics is that students utilise the power of technology to find or generate mathematics that would be time-consuming or difficult to create by ‘hand’, and once they have done so they use their mathematical skills to explain the results obtained.

For example, in the number theory topic students work on questions such as that in Figure 1.



(i) Write a program to solve the congruence $x^2 \equiv 0 \pmod{n}$.

(ii) Use your program to find the solutions to:

(a) $x^2 \equiv 0 \pmod{101}$
(b) $x^2 \equiv 0 \pmod{112}$
(c) $x^2 \equiv 0 \pmod{1009}$

(iii) Show that if p is prime then $x = 0$ is the only solution to $x^2 \equiv 0 \pmod{p}$.

Figure 1. Example number theory question from FPT.

Technology is used here to systematically work through a large number of cases but, once the results have been obtained, students are expected to give a rigorous mathematical reason for the statement in part (iii). In FPT the emphasis is on students having an understanding of the mathematics and for the technology to support them in this; the questions are intended to model the practice of searching a large number of cases but then being able to justify and explain any results.

In FPT all the programs are written in Python. However, this is not a programming course; the only expectation of the students is that they can use the mathematical functions built-in to Python with the ‘for’ and ‘if’ commands. They are then expected to be able to write short programs to perform exhaustive (or brute-force) searches. Number theory is an ideal topic for this as many of the problems relate to positive integers which are straightforward to generate with the `for` command. This is also a valuable topic as students only real experience of number theory otherwise is a brief introduction to the concept of prime numbers lower down the school. Many A level Further Mathematics students will be considering progressing to a degree in Mathematics. Studying number theory in A level Further Mathematics provides an opportunity for them to meet a topic that more resembles undergraduate mathematics than much of A level Mathematics.

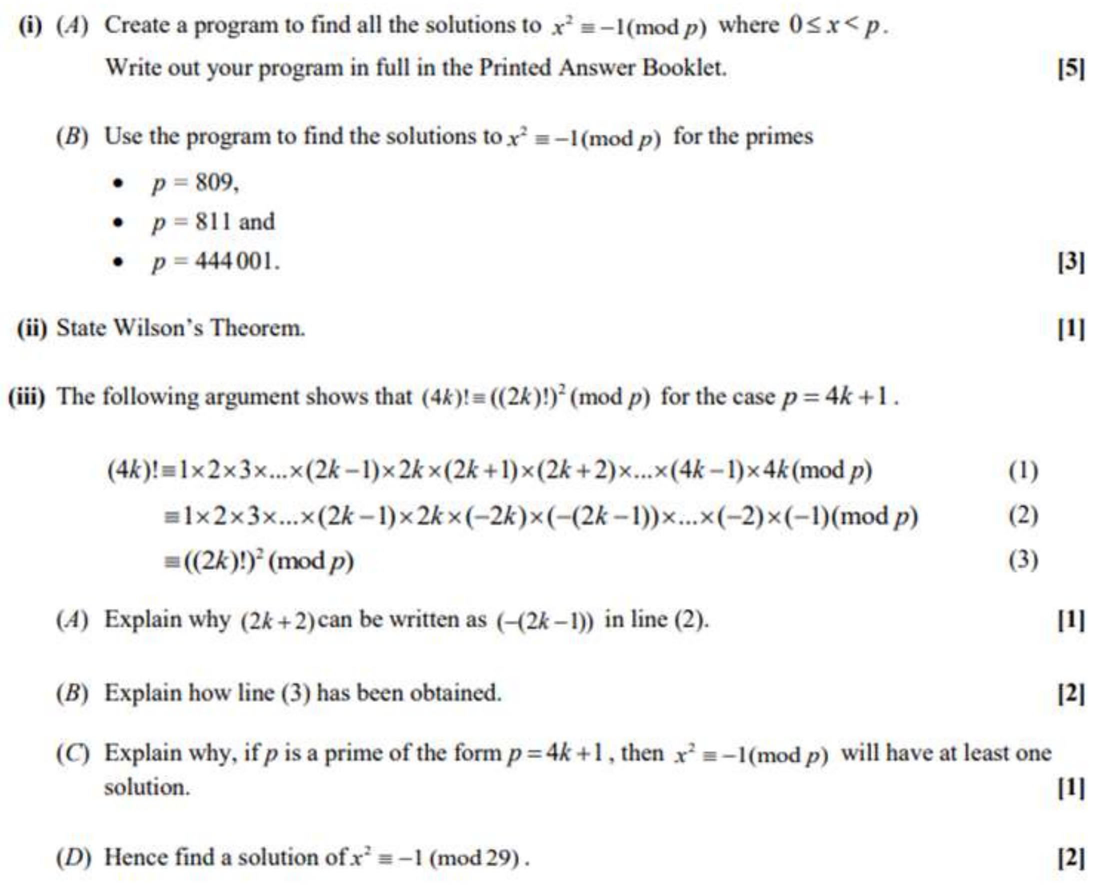
An example question from the specimen paper is shown in Figure 2. This demonstrates the standard students are expected to reach by the time they take their examinations. The full specimen paper and other related information on the FPT component of Further Mathematics can be found on the MEI website: www.mei.org.uk/fpt

3. FPT in the new ‘2017’ curriculum

As stated in section 1, for the new 2017 specifications A level Further Mathematics has some compulsory content. More specifically, 50% is a prescribed ‘pure maths core’. For the remaining 50% of the content, different options are available. These options vary between awarding organisations (AQA/Edexcel/OCR/OCR (MEI)) and may include mechanics, statistics, discrete/decision maths and additional pure maths. For the OCR (MEI) specification FPT makes up 16⅔% of A Level Further Mathematics content.

OCR (MEI) A level Further Mathematics also allows students to take additional options, with the best scores contributing to their A level grade, so FPT could be offered to students as a useful additional option, without committing to it being part of their overall mark.

One of the key points to emphasise is that even within the parameters required for devising a specification from 2017, MEI was able to develop an innovative curriculum involving aspects like FPT. An equivalent to FPT is not found in any of the other specifications.


(i) (*A*) Create a program to find all the solutions to $x^2 \equiv -1 \pmod p$ where $0 \le x < p$.
Write out your program in full in the Printed Answer Booklet. **[5]**

(*B*) Use the program to find the solutions to $x^2 \equiv -1 \pmod p$ for the primes

- $p = 809$,
- $p = 811$ and
- $p = 444\,001$. **[3]**

(ii) State Wilson's Theorem. **[1]**

(iii) The following argument shows that $(4k)! \equiv ((2k)!)^2 \pmod p$ for the case $p = 4k+1$.

$$\begin{aligned}(4k)! &\equiv 1\times 2\times 3\times \ldots \times (2k-1)\times 2k\times (2k+1)\times (2k+2)\times \ldots \times (4k-1)\times 4k \pmod p & (1)\\ &\equiv 1\times 2\times 3\times \ldots \times (2k-1)\times 2k\times (-2k)\times (-(2k-1))\times \ldots \times (-2)\times (-1) \pmod p & (2)\\ &\equiv ((2k)!)^2 \pmod p & (3)\end{aligned}$$

(*A*) Explain why $(2k+2)$ can be written as $(-(2k-1))$ in line (2). **[1]**

(*B*) Explain how line (3) has been obtained. **[2]**

(*C*) Explain why, if p is a prime of the form $p = 4k+1$, then $x^2 \equiv -1 \pmod p$ will have at least one solution. **[1]**

(*D*) Hence find a solution of $x^2 \equiv -1 \pmod{29}$. **[2]**

Figure 2. Example question from the FPT specimen paper.

The continuation of having an FPT component, when the new curriculum changed in 2017, also aligns with several of the recommendations of the Smith review into post-16 mathematics (Department for Education, 2017). Smith highlighted use of technology as a key area to maintain pace with the changing world and for teaching and the wider workforce. MEI will continue to innovate in mathematics education to support education and industry in this area.

4. References

Baldwin, C. and Lee, S., 2017. Exploring the new AS and A levels in Mathematics and Further Mathematics. *Mathematics Today*, 53(4), pp.156-158.

Department for Education, 2017. *Report of Professor Sir Adrian Smith's review of post-16 mathematics*. Available at: https://www.gov.uk/government/publications/smith-review-of-post-16-maths-report-andgovernment-response [Accessed 20 February 2020].

Glaister, P., 2017. AS and A levels in Mathematics and Further Mathematics are changing - are you ready? *MSOR Connections*, 15(7), pp.14-27. https://doi.org/10.21100/msor.v15i3.508.

Lee, S. and Button, T., 2013. Moving with the Times - A New A level Further Mathematics Unit: Further Pure Mathematics with Technology (FPT). *MSOR Connections*, 13(2), pp.10-11.

CASE STUDY

Automated assessment in a programming course for mathematicians

Henning Bostelmann, Department of Mathematics, University of York, York, UK.
Email: henning.bostelmann@york.ac.uk. https://orcid.org/0000-0002-0233-2928.

Abstract

The paper reports on a programming course for undergraduate Mathematics students in their 2nd year, with some parts compulsory for single-subject students. Assessment takes the form of several programming projects. Formative feedback as well as summative assessment is aided by automated unit tests, which allow for rapid and consistent marking, while focussing marker's time on students who require the most help.

Keywords: programming, automated assessment, unit tests, Java.

1. Introduction

Computer Science originates in Mathematics; computers are based on mathematical rules and, early on, programming was seen as a mathematical activity (Dijkstra 1974). In fact, fundamental mathematical concepts like sets, functions, their domains and codomains, have their counterpart in modern programming languages. Vice versa, computer technology plays a central role for Mathematics, its applications and its importance in society; Jaffe (1984) wrote that *"no reflection of mathematics about us is more striking than the omnipresent computer",* and this is even more valid in today's environment.

In that light, it is the author's opinion that in undergraduate Mathematics, the teaching of computer programming should be treated on equal footing to other core mathematical subjects, both in importance and level; though not everyone agrees.[1]

Certainly, there are substantial differences to usual Mathematics teaching. For one, the UK admissions process filters applicants by their mathematical abilities, but not their programming experience; hence a wide range of backgrounds needs to be catered for.

More importantly, in feedback and marking on programming tasks, an even more prominent focus than usual needs to be put on *outcomes:* computer programs need to produce the *correct* result, exactly adhering to the specified problem, and unlike perhaps in usual Mathematics assessment, there is typically no meaningful partially correct solution, unless it implements partial functionality. Without doubt, writing well-structured and well-documented programs is important (*"A programmer is ideally an essayist who works with traditional aesthetic and literary forms as well as mathematical concepts [...]"* – Knuth 1996, p.2); but first and foremost, students as well as teachers need to verify whether code works correctly. To that end, they need to *test* it with input data; just reading the source

[1] *"I never wanted to study coding and won't need it in my future profession so I don't see the practicality and relevance of it. If I wanted to learn about coding, I would have studied computer science - this is a waste of my time."* – Anonymous student feedback 2018/19.

code is not a reliable way of verification. However, taking this in earnest for marking involves a large amount of tedious manual work, and might therefore be neglected for time reasons.

In the present case study, we report on a programming course for mathematics students in which feedback and marking is assisted with automated methods, more technically, *automated unit tests*. These are used both for giving rapid feedback to students in computer practicals, and for summative assessment of programming projects.

They also aid consistency of marking and remove bias, while focusing marker's time on important tasks such as written feedback.

2. Background

The module in question is an introductory programming course for Mathematics students in the second year. It has been taught, in slightly varying forms, at the University of York since the academic year 2013/14, with the author as module leader.

2.1. Module design

The 10 credit module is intended for single subject Mathematics students in their 2nd year, students on some joint programmes, as well as Natural Sciences students who may take it as an elective. It has no prerequisites beyond Calculus and Algebra at first-year level; in particular, no knowledge of programming is assumed, even if a proportion of the audience has varying levels of experience with programming in some context.

The syllabus is split into a basic part, introducing procedural programming (variables, expressions and assignments; data types, including floating point numbers; loops and conditional structures; functions; arrays; character strings; input/output), and an advanced part, including fundamentals of object-oriented programming (dynamic methods and inheritance). These are presented along *applications* from Mathematics, such as approximation algorithms, which are not systematically taught (beyond a brief discussion of roundoff errors in floating point arithmetic), but presented as examples in exercises or lectures, without formal justification. Besides programming techniques, the module also aims to teach associated skills, including documentation.

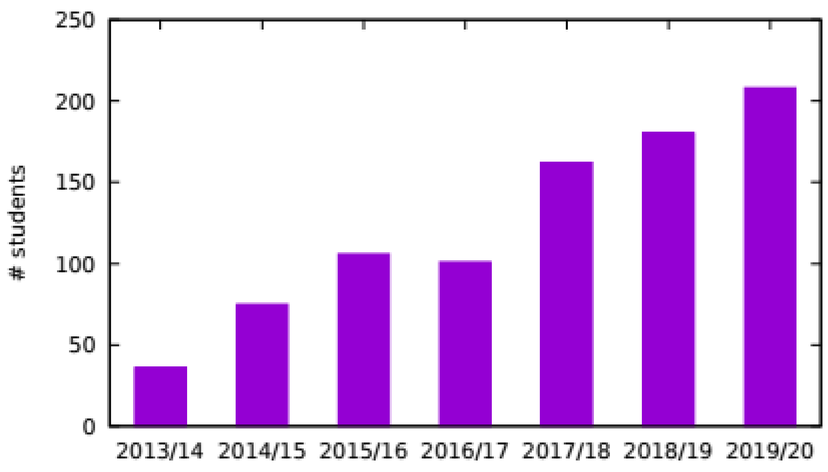


Figure 1: Participant numbers by academic year; until 2016/17: optional module 'Programming and Scientific Computing', from 2017/18: compulsory module 'Mathematical Skills 2'.

Until the year 2016/17, all of these topics were taught as one optional module ‘Programming and Scientific Computing’. From 2017/18 on, the content was incorporated into a new compulsory module ‘Mathematical Skills 2: Programming and Recent Advances’: Here students are taught the basic parts of the programming syllabus (approximately 2/3 of the material) during the Autumn term; in the Spring term, students can choose between either the advanced programming syllabus or an essay topic in Pure Mathematics, Applied Mathematics or Statistics. (These choices will not be discussed further in this article.)

Participant numbers have been rising year on year (Fig. 1) and now exceed 200 students.

2.2. Technical setup

The module aims to teach foundational programming techniques, not (only) for use in mathematical applications, but as a general employability skill. To that end, a general purpose programming language is used, rather than an application-, product-, or vendor-specific system, which should demonstrate all conceptual aspects of a modern programming language (the author includes type strictness here). It is methodologically difficult to determine which languages are most frequently used in practice, in particular in an industry setting; that said, existing surveys indicate (TIOBE, 2020; Bissyandé et al., 2013) that languages of the C family (including C++, Java and vendor-specific derivatives) continue to be used widely, if not overwhelmingly. Of these, Java appears to be the best suited in a beginner’s teaching setting.

As a development environment for Java, the module uses BlueJ, which was developed specifically for education purposes. In a comparatively simple user interface, BlueJ allows students to easily invoke individual functions of their programs, as well as offering a built-in debugger (also usable for demonstrations in lectures) and integrated unit test tools, which will become important below.

While BlueJ was originally intended for an ‘Objects First’ approach (Barnes and Kölling, 2016), the module uses it for a more traditional procedural approach for its basic part, along the first chapters of Nielsen (2009), with object-oriented techniques introduced only later.

Beyond the standard installation of BlueJ, the module uses – as an example for an external third-party library – the Apache Commons Math library (Apache 2020).

Teaching materials are provided to students via a Moodle-based VLE, which is also used for some randomised multiple choice quizzes (cf. Sec. 4).

2.3. Automated assessment

As a particular feature, and focus for this case study, the module aims to use automated assessment methods for rapid feedback to students as well as for summative assessment. Specifically, teacher-provided automated unit tests are employed for this purpose.

Unit tests are short pieces of program that run the developer’s code with certain input data, and compare its output to expected values. They are a longstanding and commonly used tool in software development (see, e.g., Runeson 2006). Here we use the same techniques for checking that a student’s work conforms to the specification given in an exercise description.

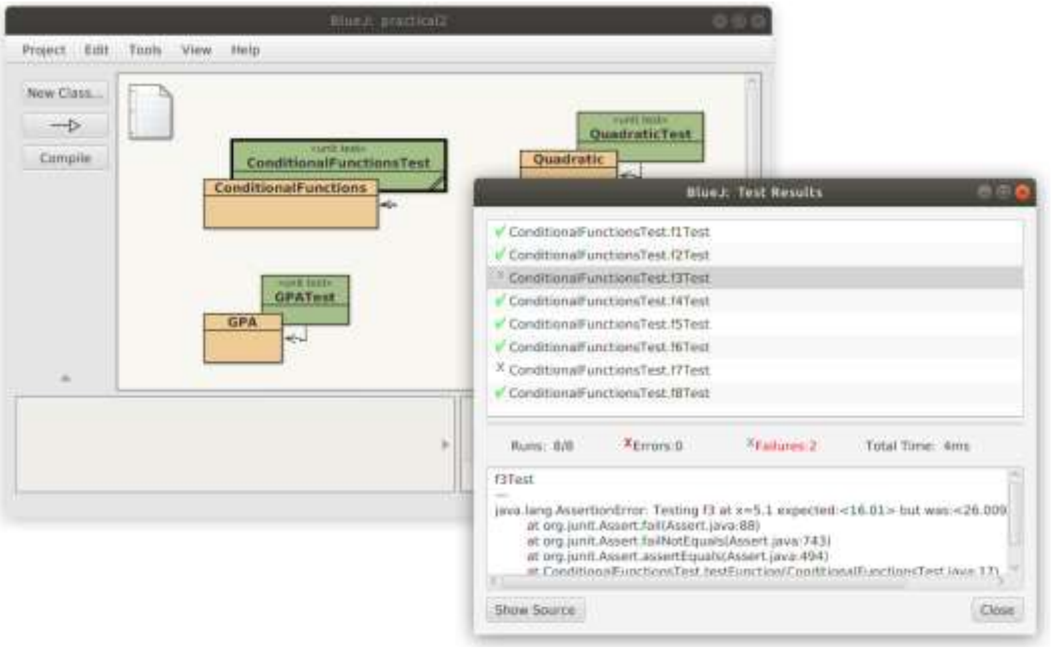
This allows student as well as teachers to verify rapidly whether a student’s work is functionally correct. It is a common misconception that programming work (even of a simple kind) can reliably be checked by reading its source code – even missing out on small errors might make the reader assume that the program is ‘correct’, whereas it actually never performs the desired functionality. In other words, student work (as any other program) needs to be *tested* with relevant input data, rather than cross-read by the teacher. Unit tests allow us to do this efficiently, freeing up teacher's time for other important aspects of the feedback process. Usage of these tests in formative and summative assignments is further described in Sec. 3 and 4 respectively.

Note that all unit test code in the module is provided by the *teacher* – students are not required to write unit tests or to understand the program code underlying them, they simply invoke them from a graphical user interface. Test code uses the unit test framework JUnit 4 (JUnit 2020), which is embedded into BlueJ.

3. Teaching and formative assessment

Teaching is centred around 9 computer practical (6 for the basic and 3 for the advanced part). Each practical is supported by 2 lectures which aim to introduce the relevant programming techniques and demonstrate examples.

Each practical comes with an exercise sheet that is released to students a few days in advance of the session, and a corresponding code template. During the session, students work independently on the exercises, but with a teacher present for help; the student/teacher ratio is approximately 15:1.



```
✓ ConditionalFunctionsTest.f1Test
✓ ConditionalFunctionsTest.f2Test
✗ ConditionalFunctionsTest.f3Test
✓ ConditionalFunctionsTest.f4Test
✓ ConditionalFunctionsTest.f5Test
✓ ConditionalFunctionsTest.f6Test
✗ ConditionalFunctionsTest.f7Test
✓ ConditionalFunctionsTest.f8Test

Runs: 8/8     ✗Errors: 0     ✗Failures: 2     Total Time: 4ms

f3Test
---
java.lang.AssertionError: Testing f3 at x=5.1 expected:<16.01> but was:<26.009
        at org.junit.Assert.fail(Assert.java:88)
        at org.junit.Assert.failNotEquals(Assert.java:743)
        at org.junit.Assert.assertEquals(Assert.java:494)
        at ConditionalFunctionsTest.testFunction(ConditionalFunctionsTest.java:17)
```

Figure 2: Unit tests within the practical materials. Screenshot from BlueJ

Unit tests are used in the practicals for rapid feedback. That is, unit tests are supplied with the code template for each exercise. Students will write code for the relevant exercise, and are requested to first run their code with relevant input data to verify that it works correctly. Once completed, they run the unit test and are presented with a 'traffic lights' result (Fig. 2). If the unit test passes, they continue to the next exercise; otherwise they can ask a teacher for assistance. (Failed unit tests will display a brief explanatory message, but this is often not detailed enough for students to localise the problem.)

Programming to pass these unit tests does require students to stick to the specifications on the exercise sheet very strictly, in particular to use the correct signatures (function or method names, input and output parameters and their data types), if not already given in the code template. This is in fact an intended learning outcome: in any collaborative programming setting – as students would encounter it in real-world applications – sticking to an agreed programming interface is crucial.

Each exercise sheet contains more exercises than can be solved in one hour (except by the most experienced students); the remaining exercises are left as homework, with no hand-in, but with feedback available via unit tests, and questions answered in the following practical session.

4. Summative assessment

The module is assessed on open assessments (Table 1), mainly consisting of 3 programming projects. These projects are marked semi-automatically; see Sec. 4.1 for details. Since the number of students on the module makes it impossible to set individual project topics, collusion may present an issue; we discuss this in Sec. 4.2.

Assessment component	**time to work on assignment**	**weight in module**	**length (LOC)**	**proportion of automated marks**	**time from deadline to mark release**	**manual marking time per submission**
Programming project 1	1 week	5%	50-100	100%	1 day	n/a
Programming project 2	6 weeks	25%	200	66%	3 weeks	20 min
Programming project 3	7 weeks	50%	300	50%	3 weeks	30 min
Online quizzes	1 week	15%		100%	immediate	n/a
Careers exercise	2 weeks	5%				

Table 1: Assessment components in Mathematical Skills 2, for students who choose the advanced programming part. Length of project given in Lines of Code (LOC) excluding documentation. All numbers except mark weights are approximate and may vary between years.

In addition to the projects, assessment includes a number of online quizzes based on multiple choice questions drawn from a random pool, which are presented and automatically marked via the Moodle VLE, and which students complete in their own time. These should be seen mostly as reading comprehension tests: they can be answered from absorbing the lecture material, without actually writing a program. The intention is to ease beginner students into the topic and allow them to collect marks for basic understanding. A careers exercise, which also counts towards the module mark, is not further discussed here.

4.1. Programming projects

In setting and marking the programming projects, assessment must be divided into two main parts: *functional* aspects (i.e., whether the code works correctly to specification) and *non-functional* aspects (whether the code is easy to read, well-structured and well-documented). In both aspects, student numbers require a distributed approach to marking, raising potential issues with marking consistency.

In all projects, functional aspects are marked with the aid of unit tests, which are prepared by the examiner in advance, but are not released to students. Student submissions are first verified against these unit tests, and a mark as well as an error report automatically generated. Marks are awarded entirely on the criterion whether the unit tests pass (typically 1 mark per test). The automated score for the functional part is only modified by the lead marker, and only under strict conditions (typically, when an entire part of the project fails to work due to a minor deviation, such as a mis-named function).

This semi-automatic marking process can be used in two ways: First, it is possible to set rapid feedback assignments (Project 1, cf. Table 1) which are marked *exclusively* on functional aspects, and almost fully automatically. Marks and automated error reports can then be released as little as 1 day after the deadline; only submissions with particularly low scores receive separate written feedback by the examiner. Verbal feedback to all students who request it is then given in the following practical session.

Second, for longer projects (Project 2 and 3), a distributed marking process is used: Automated reports are generated and shared with 3-4 markers; their role is then to determine *why* (not whether) the code fails to work, and to write corresponding feedback to students. This guarantees objectivity for this part of the assessment: the score is not subject to an individual marker's decision. It also allows markers to spend more time on feedback (rather than manual testing), and reduces possible oversights. In addition, markers assign scores for non-functional aspects along a structured marking guide and with brief per-item feedback; these items relate to the overall code structure, specific aspects of the code (e.g., expected function calls), adherence to code style conventions, and completeness of the documentation – students are typically asked to add Javadoc comments to their code.

We found the automated marking process to work smoothly in general, though some submissions need manual fixing – for example, where students use an incorrect directory structure in their submitted code. These errors typically occur for 1-2% of students and are corrected by the lead marker without penalty.

Use of automated marking in this way requires some care, because students can easily fail a large number of these tests by small omissions that may not immediately be obvious. Exact adherence to the given specification (in particular signatures) is required for tests to yield marks. To some extent, this is only a reflection of the reality of the subject: in software projects, specifications and conventions must strictly be followed to arrive at a working product.

On the other hand, some mitigation against such 'catastrophic failures' needs to be provided. To that end, the code template for every project contains a single unit test (the 'declaration test') that verifies not the functionality, but rather the function declaration and signatures in the student's code, using the Java Reflection API. Students are advised to use this test before submission to verify their basic code structure and fix any discrepancies.

4.2. Academic integrity

Project-based open assessment is an appropriate format for this module, since it comes fairly close to a realistic programming setting. While assessment by closed exam would be possible (and is sometimes used in similar situations, often with restriction on Internet access and availability of other resources), it has severe drawbacks: First, because of the limited time available, only the most elementary question can be asked, and aspects e.g. of code structure and documentation need to be dropped; second, it is a highly artificial setting which does not occur in real-world applications – no programmer would, in practice, do their work without Internet access.

However, since student numbers prohibit the setting of individual project topics, plagiarism and collusion between students is a significant concern, as probably with most other open assessments in Mathematics. In fact, it is quite common to see some students closely following their peer's solutions, with changes only in naming of variables, whitespace, etc. While rare, it has also occurred that students submitted almost literal copies of each other's work, sometimes in the form of binary identical files.

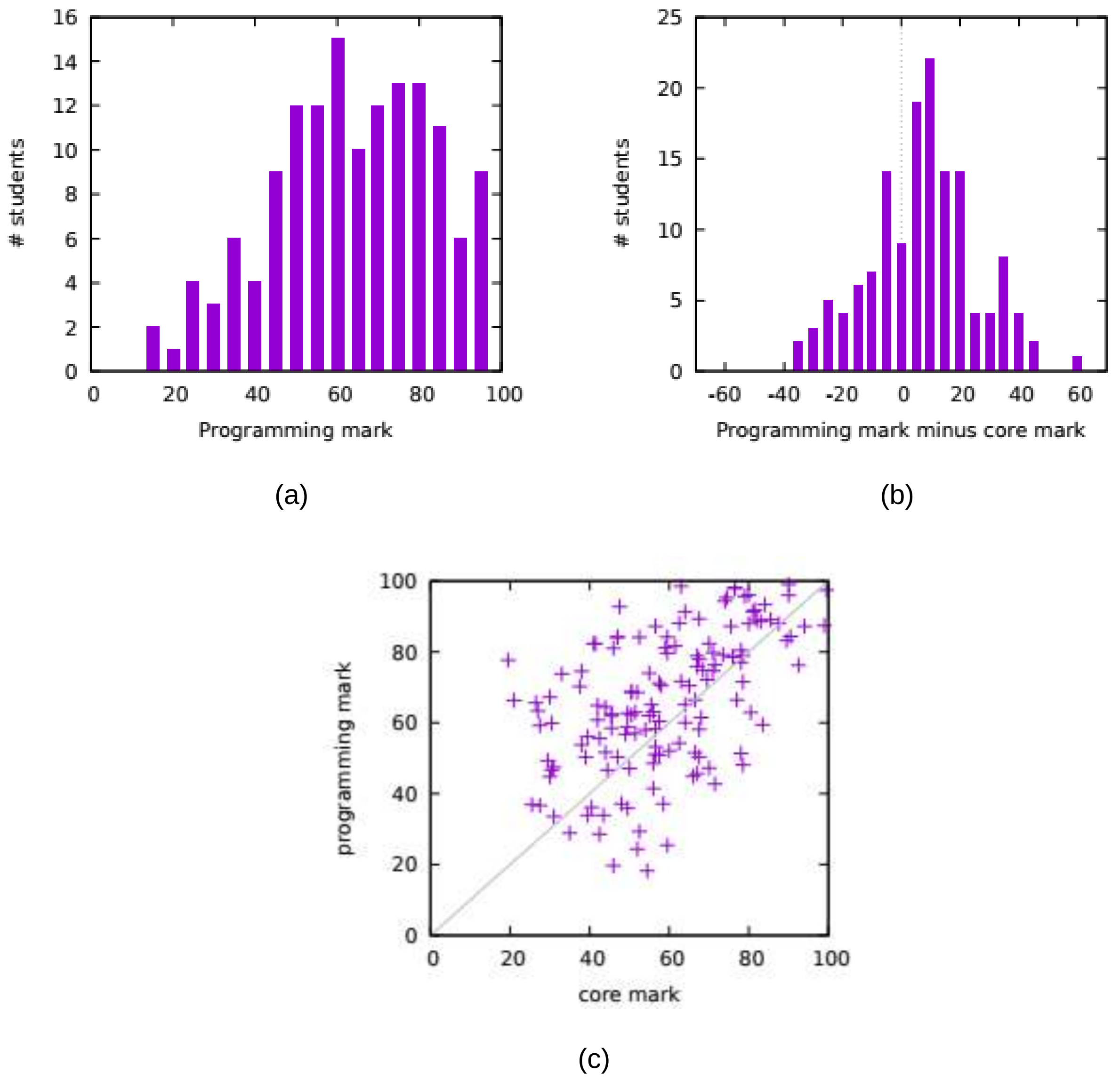


Figure 3: Programming marks vs. core marks in 2018/19. (a) Frequency of marks for the programming components; (b) frequency of differences between programming and core marks; (c) scatter plot of individual student's results.

In order to deal with this situation, marking of projects is assisted by an automatic similarity check. Specifically, we make use of the software JPlag (KIT, 2020; Prechelt et al., 2002) which is able to highlights similarities in code between student's submissions, ignoring trivial changes such as whitespace, comments, renaming of identifiers, and swapping of lines. Similarities identified by the tool are then investigated manually. We typically found relatively high numbers of similarities in Project 1 (sometimes affecting more than 5% of the submissions in various clusters), which are dealt with by means of mark reductions. After that, collusion cases in projects 2 and 3 are rare.

5. Student performance

Since programming appears to require somewhat different skills than typical Mathematics modules, and the module counts towards the degree classification, it is fair to ask how student results on the module compare with those in more traditional subjects.

To that end, we compare summative marks from the compulsory, ‘basic’ programming component with exam results from two other modules that most participants take in the same term, Vector Calculus and Linear Algebra; the average of these two will be referred to as the ‘core’ mark. Only students which took all three modules are included in the comparison; also, students who missed any of the assessments, or who had resits ‘as if for the first time’ approved for any of them, have been excluded from the analysis.

The results for 2018/19 are shown in Fig. 3. One notes that the mark distribution for the programming part alone (Fig. 3a) is within usual expectations, though possibly somewhat high in the 80-100 range, as is typically seen with coursework-based assessments. The differences between programming marks and core marks (Fig. 3b) show that there is a good correlation between general Mathematics performance and marks in programming, while there are a few wide outliers. This is confirmed by the scatter plot in Fig. 3c. In short, while there are some Mathematics students with good marks that struggle with programming tasks (and vice versa), these seem to be an exception rather than the rule.

6. Conclusions

The module discussed in this paper demonstrates that it is possible to integrate a generic programming course as a compulsory element into a Mathematics programme. It scales to the size of undergraduate cohorts, with student performance in line with expectations. Automated unit tests, when suitably set up, help with providing rapid feedback, ensuring consistency of marking, and making prudent use of staff resources.

7. References

Apache Software Foundation, 2020. *Commons Math library*. Available at: http://commons.apache.org/proper/commons-math/ [Accessed 24 February 2020].

Barnes, D.J. and Kölling, M., 2016. *Objects first with Java: a practical introduction using BlueJ*, 6th edition. London: Pearson.

Bissyandé, T.F., Thung, F., Lo, D., Jiang, L. and Réveillère, L., 2013. Popularity, Interoperability, and Impact of Programming Languages in 100,000 Open Source Projects, *IEEE 37th Annual Computer Software and Applications Conference, Kyoto*. pp. 303-312. http://doi.org/10.1109/COMPSAC.2013.55.

Dijkstra, E.W., 1974. Programming as a Discipline of Mathematical Nature. *The American Mathematical Monthly,* 81(6), pp.608-612. http://doi.org/10.1080/00029890.1974.11993624.

Jaffe, A., 1984. Ordering the Universe: The Role of Mathematics. *SIAM Review,* 26(4), pp.473-500.

JUnit project team, 2020. *JUnit 4*. Available at: https://junit.org/junit4/ [Accessed 19 February 2020].

Karlsruhe Institute of Technology (KIT), 2020. *JPlag - detecting software plagiarism.* Available at: https://jplag.ipd.kit.edu/ [Accessed 10 February 2020].

Knuth, D.E., 1996. *Selected Papers on Computer Science*. Chicago: University of Chicago Press.

Nielsen, F., 2009. *A Concise and Practical Introduction to Programming Algorithms in Java.* Springer: London. http://doi.org/10.1007/978-1-84882-339-6.

Prechelt, L., Malpohl, G. and Philippsen, M., 2002. Finding plagiarisms among a set of programs with JPlag. *Journal of Universal Computer Science*, 8(11), pp.1016-1038. http://doi.org/10.3217/jucs-008-11-1016.

Runeson, P., 2006. A survey of unit testing practices. *IEEE Software,* 23(4), pp.22-29. http://doi.org/10.1109/MS.2006.91.

TIOBE Software BV, 2020. TIOBE Index. Available at: https://www.tiobe.com/tiobe-index/ [Accessed 11 February 2020].

OPINION

Automated Assessment of Python Code

Sam Morley, School of Mathematics, University of East Anglia, Norwich, UK. Email: Sam.Morley@uea.ac.uk. https://orcid.org/0000-0001-5971-7418.

Keywords: python, assessment, code quality.

1. Introduction

Python is a high-level programming language that is interpreted and executed by a runtime, most commonly CPython. It is fast becoming one of the most popular languages due to its flexibility and interoperability with optimised tooling and libraries. Moreover, Python is syntactically simple, which makes it ideal as a programming language for teaching and learning as a first language. The fact that Python is interpreted and executed together at execution time gives Python the ability to perform deep introspection, which also makes it an excellent language for automated assessment.

The marking criteria for code can usually be categorised as either *style* or *operation*. For beginner programmers, operation – whether the code performs the intended task correctly – is arguably more important. As experience grows, it becomes increasingly important to write well-structured and idiomatic code that is easy to read (style). More advanced programmers might also consider how robust their code is; how it handles expected or unexpected errors. For introductory programming courses, assessments should place greater emphasis on operation and less on style. However, including marking criteria for coding style will help encourage good coding practice, so it might be beneficial to include some style criteria.

Automated assessment offers various advantages over manual assessment, where an assessor would run the students' code and check the output by hand. Running a test suite over a large collection of student's code by hand can be extremely time consuming, whereas an automatic tool can run tests in the background. Automated test suites can also run in parallel, which can lead to a greatly decreased total testing time. However, automated tests are more time consuming to set up.

The author is currently, at time of writing, teaching Python programming as part of a first year undergraduate module, and is designing the assessment of Python code for this course. This assessment is intended to be marked automatically. In this article, we give a quick overview of some techniques, packages, and tools that may be useful for automatically assessing Python source code. At the end of the paper, we give a short example of a function that could be used to mark a script-based submission.

2. Assessing operation

Arguably the most important aspect of assessing the operation of source code is checking correctness. First and foremost, code should always produce predictable and repeatable results, given the same environment and input, which can be tested by a suite of individual *unit tests*. A unit test is usually a small function that tests a specific aspect of a program using assertions. The Python Standard Library contains the unittest module, that provides support for writing and running unit tests on a Python module (program). There are third party alternatives that offer a much more flexibility and features, the most popular of which is Pytest (2020).

Automated suites of tests provide a much more consistent approach to checking correctness than manual tests – that is, a user (assessor) manually running each module/item with given input and comparing output – and is usually much faster. On the other hand, test suites can be difficult to construct properly and require a larger investment of time to write. It might also be possible to link the running of the test suite automatically upon submission via a virtual learning environment (VLE), to provide almost real-time feedback to students, which is useful for formative work. (Integration of automated testing into deployment of code is widely used in industry, via *continuous integration* services, as a fail-safe against errors that could impact the function of the program.)

There are, however, some practical concerns about using automated testing suites to test a cohort of student source code. First, the testing utilities mentioned above rely on predictable names and locations of source files. They rely on Python's standard import mechanism to bring objects into the testing suite, and this mechanism requires hard coded names of packages/modules relative to a path recognised by Python. This is difficult to accommodate if each student's code is stored in a separate directory or named according to some identifying information. (Many VLEs will automatically *mangle* the filename of submitted files to contain key information such as the submission identifier, student ID, submission time, and original file name.) Second, these tools are aimed towards testing specific elements from a large body of code, such as single function or class. It can be difficult to adapt them to test "script-like" Python files that execute immediately at the top-level.

These practical difficulties can be overcome without too much trouble. The first problem can be solved by loading the student's code in a more controllable way. For example, we could write a small script that modifies the Python path variable to cause the import statement to look for code in a different directory for each submission. The test suite can then be executed once for each submission, controlled by the script, and we can be sure that every student's submission is tested in the same way. The second problem can be mitigated by setting questions very carefully, or writing tests that execute submission files individually and check output by inspecting the stdout/stderr streams. This can cause some problems, because of the lack of standardisation for returning information from a script. (Printing to the terminal works fine in theory, but in practice string matching can be difficult, even if the form of the printed string is known; in general, this will not be the case.)

2.1. Assessing robustness and good design

A second aspect of testing operation of a source file is testing the robustness of code. For example, we might test how a student's code reacts when an error occurs (while opening a file, for example). Code that handles such errors gracefully and potentially recover, if appropriate, might be described as *robust.* A more typical example would be to test if the student has used any type and value validation in their code to raise an appropriate exception (or assertion) if an argument is provided that is not valid.

These features might not be directly related to the execution of an element in the program, but they are certainly part of good design. Since Python is not strongly typed, we cannot rely on a compiler to check that the types provided as arguments are valid and fulfil the requirements of the function. In practice, this can lead to obscure and unexpected errors that are difficult to fix. Stressing good practice and style should be part of every course on programming, regardless of level, although it will not usually be the most critical element of assessment.

Other good design elements such as breaking repeated blocks of code into individual functions can be more difficult to test automatically, and instead will probably require somebody to read the code. Perhaps some of the tracing and debugging tools within Python (or newly introduce audit hooks, provided in CPython 3.8) could be used to check the usage of functions at runtime, but it is not obvious how to implement criteria for this task.

3. Assessing style

In addition to the assessment of operation, the code can be analysed for compliance with specific style guide, such as the official Python style guide PEP8 (van Rossum, Warsaw and Coghlan, 2013). This is usually achieved using a *linter*, which inspects the code and compares the actual code as written to some idealised version. The resulting differences are reported back to the user via the terminal, or by some other means. Popular linting programs include PyLint (Logilab, n.d.) and Flake8 (Stapleton Cordasco, 2016), although there are a large number of such programs available. These tools fall into the category of *static analysis* (analysing code without running it).

Linters can detect syntax errors and typographical errors within code before the code is executed, which makes them particularly useful as part of a build and deploy pipeline in industry (such as in a continuous integration process). Some tools can also detect code that is not *Pythonic*. This means that the programmer has not made use of Python idioms in their code. For instance, it might detect when a programmer has written code for iterating over a list using a for loop over a range and accessing each item by index, rather than leveraging the iterator protocol. While technically correct, iterating over a list by index can often lead to errors if not done correctly and is frequently slower than the iterator equivalent, particularly if the element is accessed numerous times within the body of the loop. (Each lookup incurs a function invocation rather than a simple pointer lookup. This difference is miniscule but significant in large loops.)

Generally, a linter will output a list of messages regarding issues with the code checked into the terminal, and it is up to the user to parse this list and generate a score, if necessary. The messages that are output can usually be customised to include only certain issues, such as syntax errors or convention items. It might be possible to access an application programming interface (API) for a linter and link this directly into an assessment script.

4. Example

We now give a short example function that would mark scripts submitted by students, which I supposed to plot a simple function using the Matplotlib library. We omit the code that handles finding and reading student code and the handling of edge cases for the sake of space.)

```
from unittest import mock
from io import StringIO
import matplotlib.pyplot

def mark_code(sub_code):
    patcher = mock.patch("matplotlib.pyplot", autospec=True
    mpl_mock = patcher.start()
    try:
        exec(sub_code)
    except Exception as err:
        return 0, "Your code did not execute"
    patcher.stop()
    if mpl_mock.plot.called:
       return 1, "You plotted something, well done"
    return 0, "You didn't plot anything"
```

The main step involved here are *mocking* the Matplotlib Pyplot module to test whether it was used when the submission code was executed. Once this is done, we can inspect the Mock object to see which routines have been called, in this case the plot routine. We execute the submission code inside a try block using the built in exec function. This replicates the way that Python executes code from a command line. The function returns a mark (0 or 1) and some simple feedback. In this case, the submission would get 1 mark if the plot routine was called, and 0 otherwise.

5. Conclusions

The most important part of assessing Python code written by students is testing whether the code runs and meets the requirements of the exercise. This can be achieved using tools from the software testing facilities available for Python, such as the unittest module from the Python Standard Library, or the Pytest package. However, one should not neglect the importance of good coding style when writing assessments, even if this is only used formatively. Tools such as PyLint can be used to quickly assess the style of Python code, and provide useful feedback to students on how to improve the style of their code.

6. References

Stapleton Cordasco, I., 2016. *Flake8: Your Tool For Style Guide Enforcement*. Available at: https://flake8.pycqa.org/en/latest/ [Accessed 9 July 2020].

Logilab, n.d. *Pylint*. Available at: https://pylint.org/ [Accessed 9 July 2020].

Pytest, 2020. *pytest: helps you write better programs*. Available at: https://docs.pytest.org/en/stable/ [Accessed 9 July 2020].

van Rossum, G., Warsaw, B. and Coghlan, N., 2013. *PEP 8 -- Style Guide for Python Code*. Available at: https://www.python.org/dev/peps/pep-0008/ [Accessed 9 July 2020].

CASE STUDY

Assessment of computing in the mathematics curriculum using Numbas

Chris Graham, School of Mathematics, Statistics & Physics, Newcastle University, Newcastle Upon Tyne, UK. Email: christopher.graham@ncl.ac.uk.

Abstract

This case study discusses the use of the Numbas e-assessment system to assess computing skills across several modules in a mathematics undergraduate degree programme. The modules include basic computing, quantitative analysis of data, and numerical methods. Several approaches are discussed which fit with the teaching of SPSS, R and MATLAB, including randomised data files and questions which can replicate, and therefore mark, calculations made with R data frames and numerical algorithms, such as root finding and curve fitting. In each case, Numbas offers the opportunity to automatically mark and offer immediate feedback to the student. The application of questions inside a computing module is discussed, with a positive response from students to both practice material and hybrid tests, which include some automatic marking alongside submission of the students' code for manual review. There is clear rationale for using an e-assessment system which is already familiar to students, with features such as adaptive marking and the scaffolding of questions, however limitations to the use of Numbas for this purpose are also discussed.

Keywords: computing, e-assessment, Numbas, numerical, quantitative.

1. Background

1.1. Computing in the curriculum

Modules dedicated to computer programming have been a compulsory component of the single-honours mathematics degree programme at Newcastle University since 2015. The addition of computing is in common with many other mathematics departments in the United Kingdom (Sangwin, 2017), motivated by the increasing relevance of computers in mathematical teaching and research, and in the future career prospects of undergraduate students. The curriculum has included the teaching of two programming languages, R and MATLAB (the latter being phased-out, to be replaced entirely with Python by 2022), covering basic programming, statistical analysis and numerical methods.

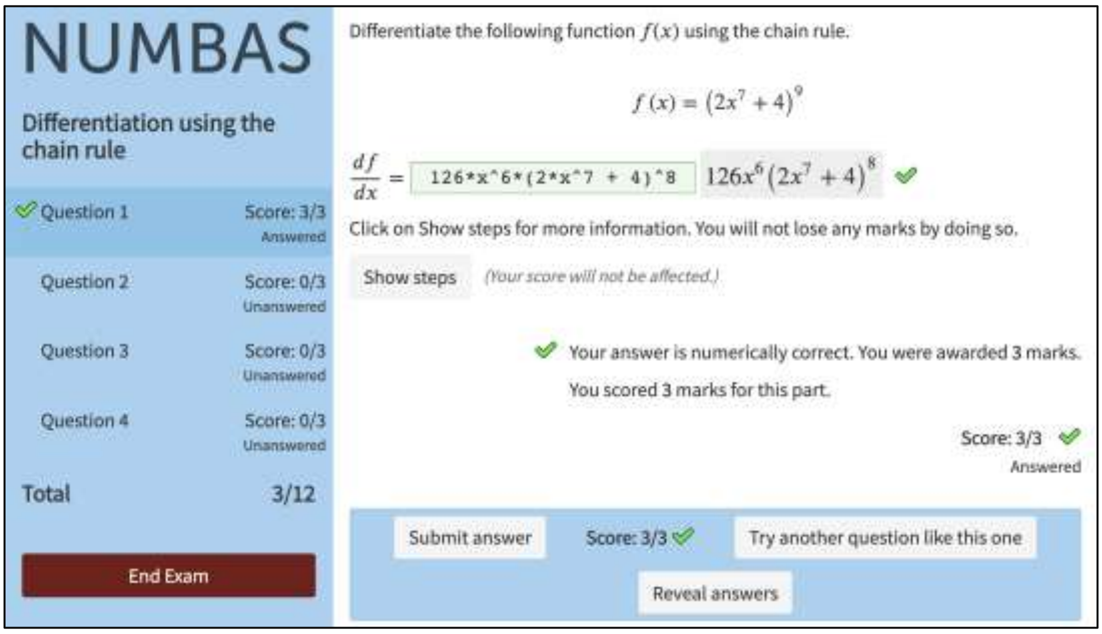
Most of the computing teaching follows the same format, with three hours of contact time each week. A one-hour lecture is used to introduce the theory behind the week's content and to work through examples. In a two-hour practical that follows, students work through a "handout" of material, which may be either a physical handout or a digital version.

1.2. The Numbas e-assessment tool

Many mathematics departments take advantage of specialist mathematical e-assessment software, such as DEWIS (Gwynllyw and Henderson, 2009), Numbas (Foster, et al, 2012) and STACK (Sangwin, 2015), to automatically mark students' work and offer immediate feedback. At Newcastle, where the software Numbas is developed, e-assessment is embedded into the mathematics degree programme across twenty-four modules.

Following the CALM model (Beevers, et al, 1988), Numbas tests can consist of several questions, each of which may contain one or more parts which assess an answer. A student answer may be a numerical value, mathematical expression (as in figure 1), or one of several other types supported, such as matrix input.

An example Numbas test


NUMBAS

Differentiation using the chain rule

✓ Question 1 Score: 3/3 Answered

Question 2 Score: 0/3 Unanswered

Question 3 Score: 0/3 Unanswered

Question 4 Score: 0/3 Unanswered

Total 3/12

End Exam

Differentiate the following function $f(x)$ using the chain rule.

$$f(x) = \left(2x^7 + 4\right)^9$$

$\frac{df}{dx} =$ `126*x^6*(2*x^7 + 4)^8` $126x^6\left(2x^7 + 4\right)^8$ ✓

Click on Show steps for more information. You will not lose any marks by doing so.

Show steps (Your score will not be affected.)

✓ Your answer is numerically correct. You were awarded 3 marks.

You scored 3 marks for this part.

Score: 3/3 ✓ Answered

Submit answer Score: 3/3 ✓ Try another question like this one

Reveal answers

Figure 2. Numbas can mark and give feedback on mathematical expressions. To the right of the student input (highlighted green to indicate a correct answer) is a preview of the student’s answer as interpreted by the software, which a student can check before submission. The settings of this test are such that the student receives immediate feedback, and has the opportunity to “Try another question like this one”, which re-randomises the variables in the question, or to “Reveal answers” to see a full solution. This question has the option to “Show steps” to see a reminder of the chain rule, which is offered without a reduction in the available marks in this case.

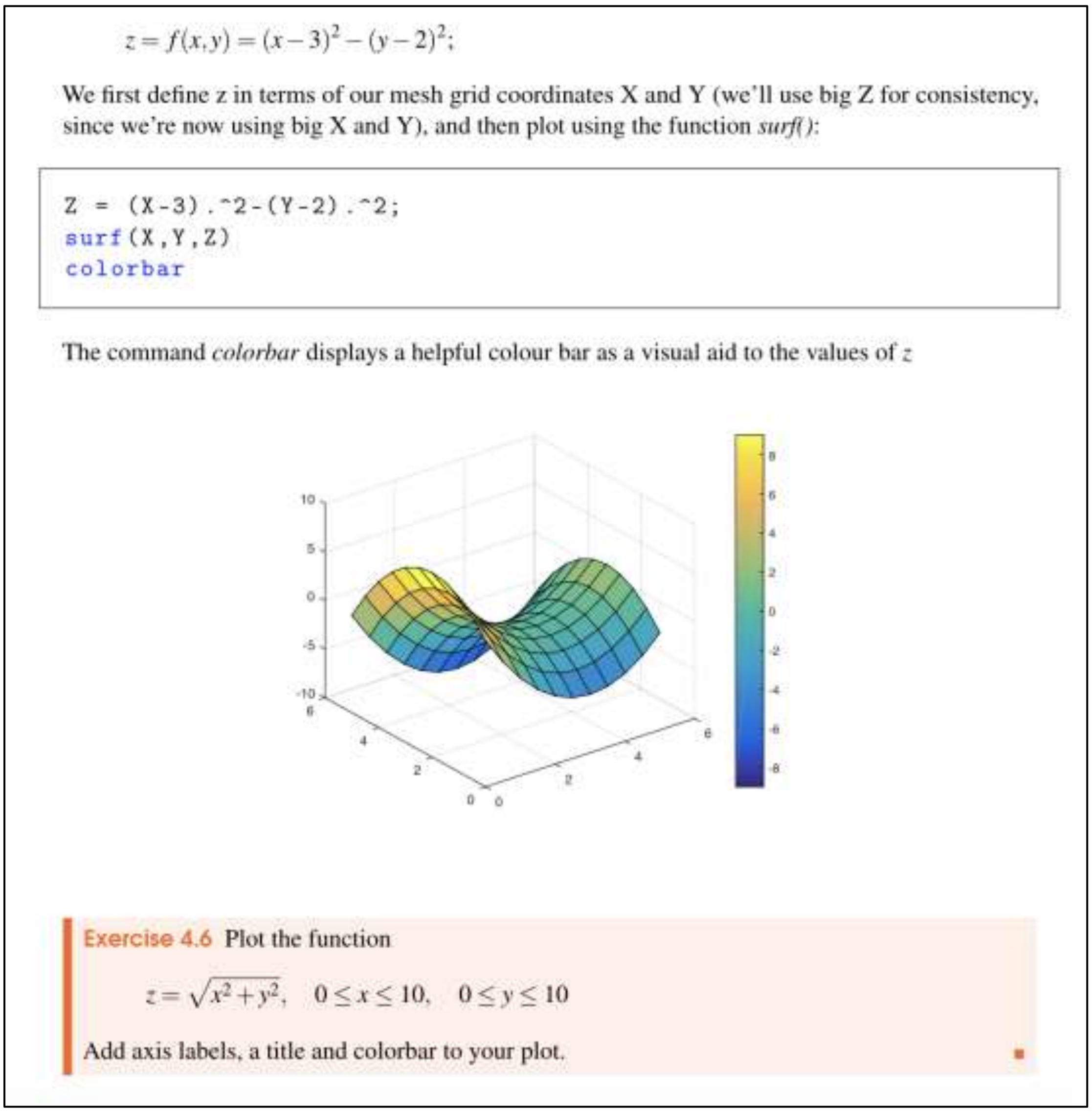
Some of the key features of Numbas were inherited from the CALM model, including advice for each question and the facility to scaffold a question into smaller steps. The variables in a Numbas question can be randomised to allow for many versions, each one with a corresponding full solution in the advice. This randomisation, alongside the immediate marking and feedback, makes the tool particularly powerful for formative use. One common complaint by students towards online assessment is its all-or-nothing marking. In response, a feature of adaptive marking was added, whereby the correct answer to a question part could be re-calculated, based on the incorrect student input to a previous part. This digitalises some of the discretion given by a human marker when awarding “error carried forward” marks. These features were important in the choice to use Numbas in computing modules.

Numbas is an open-source tool and is made available to users outside of Newcastle through a public editor hosted by MathCentre (Newcastle University, 2020). Here, teachers can create and share content, with around 8,000 items available for reuse under an open access licence, including many of the examples in this case study.

2. Motivation for using e-assessment in computing teaching

Students working through handouts in computer practicals do so at their own pace, running and investigating the commands given to them. One of the key features of the handouts is formative questions, which help to reinforce the ideas covered, or to investigate a little bit further than is covered in the given material (see figure 2). These questions are typically located at the end of a section, and students work through them as they reach them. Students are encouraged to have a go at the questions before asking for help, including using documentation and online resources. A student may request help from a postgraduate assistant, but many will suffer in silence or skip past questions if they get stuck. The lecturer may often present solutions to the questions at appropriate points during the practical, delivered by display screens and microphone to the entire class.

Example content from a computing handout



$$z = f(x,y) = (x-3)^2 - (y-2)^2;$$

We first define z in terms of our mesh grid coordinates X and Y (we'll use big Z for consistency, since we're now using big X and Y), and then plot using the function *surf()*:

```
Z = (X-3).^2-(Y-2).^2;
surf(X,Y,Z)
colorbar
```

The command *colorbar* displays a helpful colour bar as a visual aid to the values of z

10
5
0
-5
-10
6
4
2
0
0
2
4
6
8
6
4
2
0
-2
-4
-6
-8

Exercise 4.6 Plot the function

$$z = \sqrt{x^2+y^2}, \quad 0 \le x \le 10, \quad 0 \le y \le 10$$

Add axis labels, a title and colorbar to your plot.

Figure 2. An excerpt from an early version of the handouts for stage one MATLAB computing. Exercises embedded inside the handout encourage students to consolidate their learning, but when and how to provide solutions poses a problem.

Students work through computing handouts at very different rates, and the question of when to offer solutions to these questions is a difficult one. In a survey of students conducted at the end of the module in the 2019/20 academic year, 48% of 81 respondents agreed that the solutions presented by the lecturer were at the right time to help them with the handout questions, however 21% responded that these interactions were actually distracting. Students were split evenly over whether the solutions to the questions should be released immediately, or at the end of the practical.

E-assessment offers a potential solution: marking and feedback can be offered as and when a student is ready to attempt a practice question. Several tools already exist for marking computer code, including the Moodle plugin Coderunner (Lobb and Harlow, 2016) and the INGInious assessment platform (Derval, et al, 2015), which run students` code and carry out unit tests to generate marks and feedback. However, we are often interested in the answers that students can interpret from their code, rather than the code itself, and this is exactly the sort of input that Numbas handles so well: numeric values, for example from a statistical analysis, or; mathematical expressions, such as a best fit polynomial for some data. We might also be interested in breaking down questions into smaller steps or using adaptive marking to account for errors in multi-part questions, both of which are accommodated by Numbas. Since the tool is already familiar to our students, it is a logical choice to adapt it for computing questions.

3. Using Numbas to assess computing skills

Each of the examples of using Numbas to assess computing skills uses a similar flow:

- The student is presented with a question generated by Numbas, which may include some randomisation;
- The student tackles the problem on the external software, to obtain an answer;
- Returning to Numbas, the student inputs their answer. Behind the scenes, Numbas can make the same calculations as the external software, and therefore can mark the answer and offer feedback.

There is nothing particularly pioneering about this process; if you were to replace “external software” with “calculator” or “pen and paper” then this is no more than the regular workflow of using an e-assessment system. The key parts are the generation of the data and subsequent parallel analysis that Numbas can make to mark the student’s work. Note that, using these methods, Numbas does not directly mark the student’s code. This does not mean that it cannot offer advice on the code required to obtain the answer, but it relies on these skills being assessed by other means if that is desired. Section 4 describes this hybrid approach to marking for some assessments.

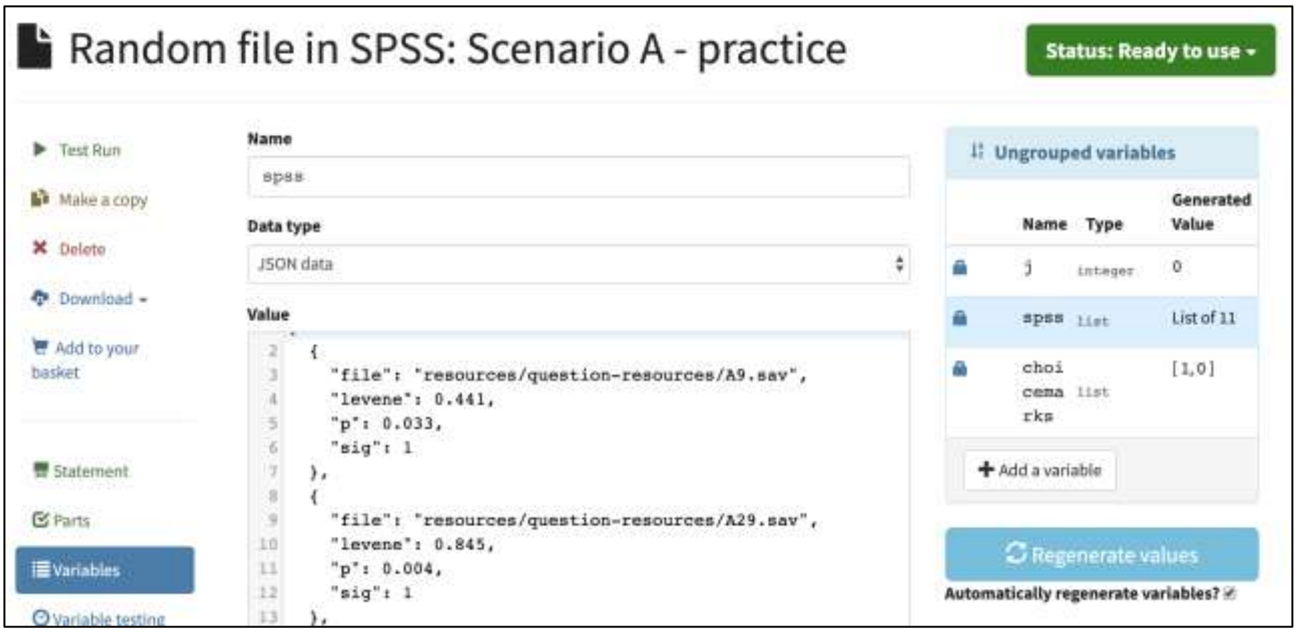
The following sections describe some of the different approaches used to develop questions:

3.1. Scenario-based questions

Assessing a student’s quantitative analysis of data inside external software is made possible by the flexibility of data types available to Numbas to store a copy of the data, including lists and arrays, and the JSON (JavaScript Object Notation) format, which is a lightweight, human-readable format for storing data.

An early example of using Numbas with external software was for a module in quantitative methods with SPSS. The question in figures 3 and 4 assesses the student’s ability to import a file into SPSS, and to perform a two-sample t-test and Levene’s test of equal variances. The student is presented with a random binary SPSS file, of the sort that they are familiar with in other parts of their teaching materials. In this case, Numbas does not directly handle the data. Rather, for each scenario (file presented to the student), the correct answers are computed in advance using SPSS, and stored in an array of scenarios in the JSON format, alongside the path to the corresponding data file (figure 3).

Numbas Editor interface for a scenario-based randomised SPSS question



Random file in SPSS: Scenario A - practice

Name: spss

Data type: JSON data

Value

```
{
   "file": "resources/question-resources/A9.sav",
   "levene": 0.441,
   "p": 0.033,
   "sig": 1
},
{
   "file": "resources/question-resources/A29.sav",
   "levene": 0.845,
   "p": 0.004,
   "sig": 1
},
```

Ungrouped variables

Name	Type	Generated Value
j	integer	0
spss	list	List of 11
choicemarks	list	[1,0]

Automatically regenerate variables? ✓

Figure 3. The variable definition in the Numbas Editor for the question illustrated in figure 4. The JSON format allows the structure of scenarios to be laid out, with each scenario containing the path to the relevant file and the correct answers associated with it.

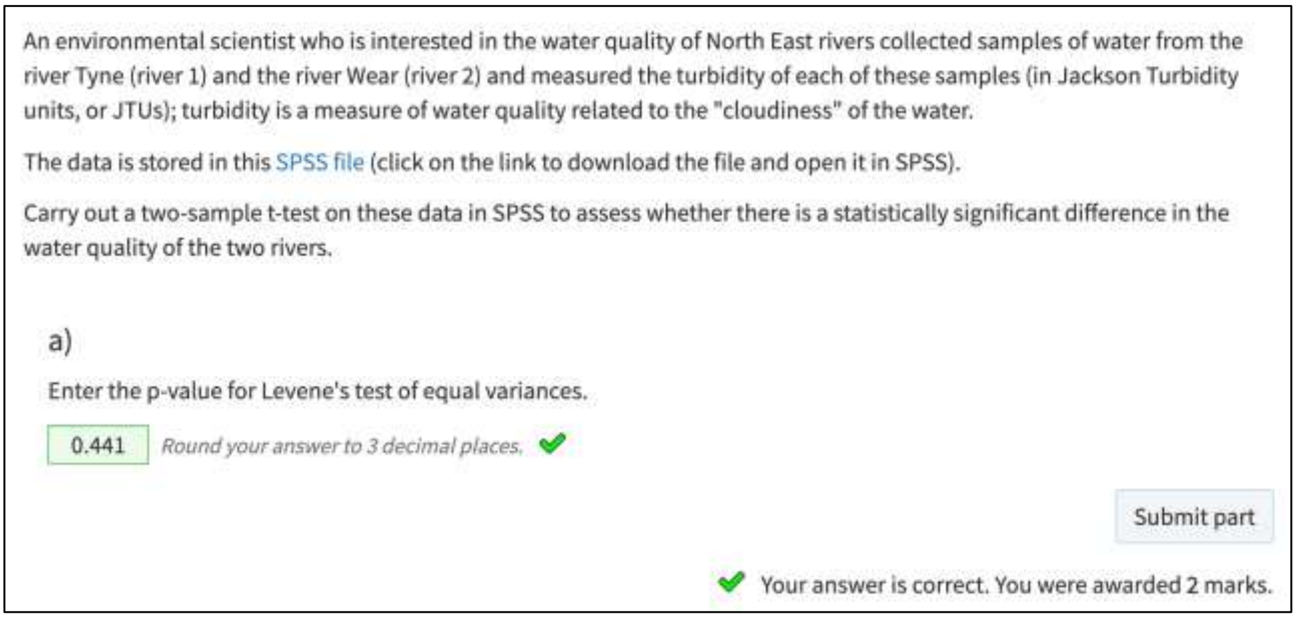
When the question is loaded (figure 4), one of the scenarios is chosen at random and the student receives a file to download. Once they have performed the analysis in SPSS, they return to Numbas to input their answers, and these are compared to the corresponding values for that scenario.

The randomisation of this question is limited by the number of scenarios prepared. It does, however, demonstrate a model for a question that is flexible to any file type or external software, and relatively straightforward to prepare in Numbas, in that it is not necessary to follow along with the calculation of answers.

3.2. Generating data files

An alternative version of the SPSS question provides a data file in CSV (comma-separated values) format, which is text-based and can be generated on-the-fly by Numbas. The Numbas Statistical Functions extension supports many functions for sampling and statistical analysis, and can be used both to generate the data (for example for a particular distribution) and to compute the correct answers. This approach is used in several questions which ask the student to input a file to R and perform queries on the imported data. A similar approach was taken to generate SPSS data and answers by embedding R code inside a DEWIS question by Weir, et al, 2017.

Numbas randomised SPSS question


An environmental scientist who is interested in the water quality of North East rivers collected samples of water from the river Tyne (river 1) and the river Wear (river 2) and measured the turbidity of each of these samples (in Jackson Turbidity units, or JTUs); turbidity is a measure of water quality related to the "cloudiness" of the water.

The data is stored in this SPSS file (click on the link to download the file and open it in SPSS).

Carry out a two-sample t-test on these data in SPSS to assess whether there is a statistically significant difference in the water quality of the two rivers.

a)

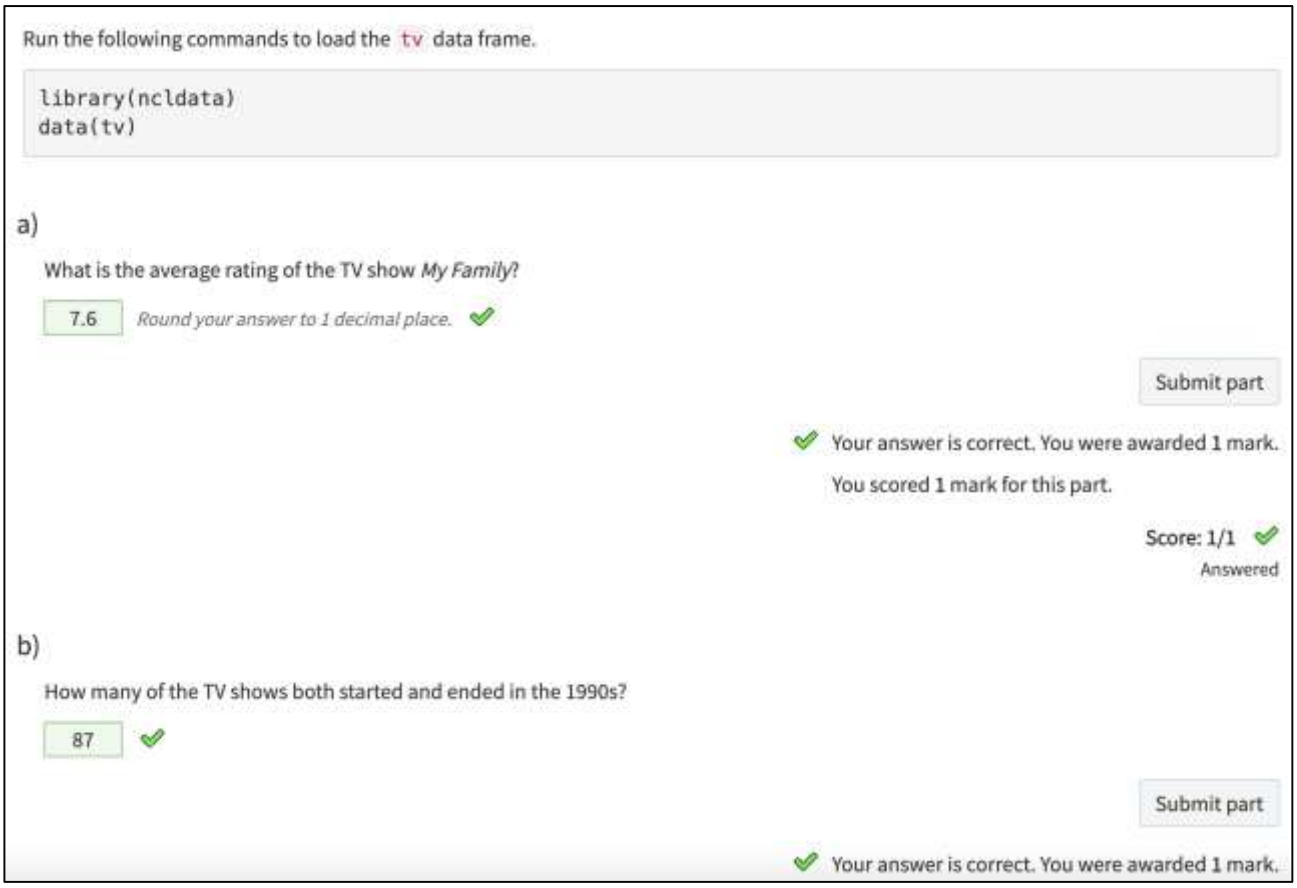
Enter the p-value for Levene's test of equal variances.

0.441 *Round your answer to 3 decimal places.* ✓

✓ Your answer is correct. You were awarded 2 marks.

Figure 4. A randomised SPSS question in which the student is presented with a random file to download. Numbas marks the student's input against the corresponding correct answer.

A randomised question using an R data frame


Run the following commands to load the `tv` data frame.

```
library(ncldata)
data(tv)
```

a)

What is the average rating of the TV show *My Family*?

7.6 *Round your answer to 1 decimal place.* ✓

✓ Your answer is correct. You were awarded 1 mark.

You scored 1 mark for this part.

Score: 1/1 ✓

Answered

b)

How many of the TV shows both started and ended in the 1990s?

87 ✓

✓ Your answer is correct. You were awarded 1 mark.

Figure 5. In this practice question, which accompanies a handout on R data frames, students are instructed to load the TV data frame. Variables are used to randomise parts of the question, in this case the show and decade.

3.3. R data frames

The material for R teaching at stage one is centred on data frames from the IMDB (Internet Movie Database) film information, which is provided through an R package (Stagg, 2019). The data set contains metadata for the films along with aggregated user rating and is, anecdotally at least, engaging for students, who are familiar with the context and interested in the content. Students learn how to query and subset the data frame, different ways to plot and present the data, and later some basic statistical analysis.

To offer practice material to students, we take advantage of the JSON format to store a copy of the data inside Numbas, in order to randomise and automatically mark questions. Figure 5 illustrates one such question.

3.4. Numerical methods

Numerical methods taught in stage two of the mathematics undergraduate degree include root finding, curve-fitting, numerical integration and solutions to ordinary differential equations. Many of the associated algorithms are well-defined and can be replicated in Numbas, which has the facility to include user-defined functions written in JavaScript.

A question might ask a student to use MATLAB's *polyfit* to fit a polynomial to a set of data points. Numbas can randomise the data itself and then replicate the functionality of the *polyfit* function to mark the student's answer. The student could then be asked to calculate the sum of the squared residuals of the fit. Since Numbas has the student's answers to the first part, it can also use adaptive marking to mark this part, as illustrated in figure 6.

4. Assessment Methodology

This section describes several different use cases for the Numbas questions in the numerical methods section of a stage 2 computing module.

4.1. Formative material

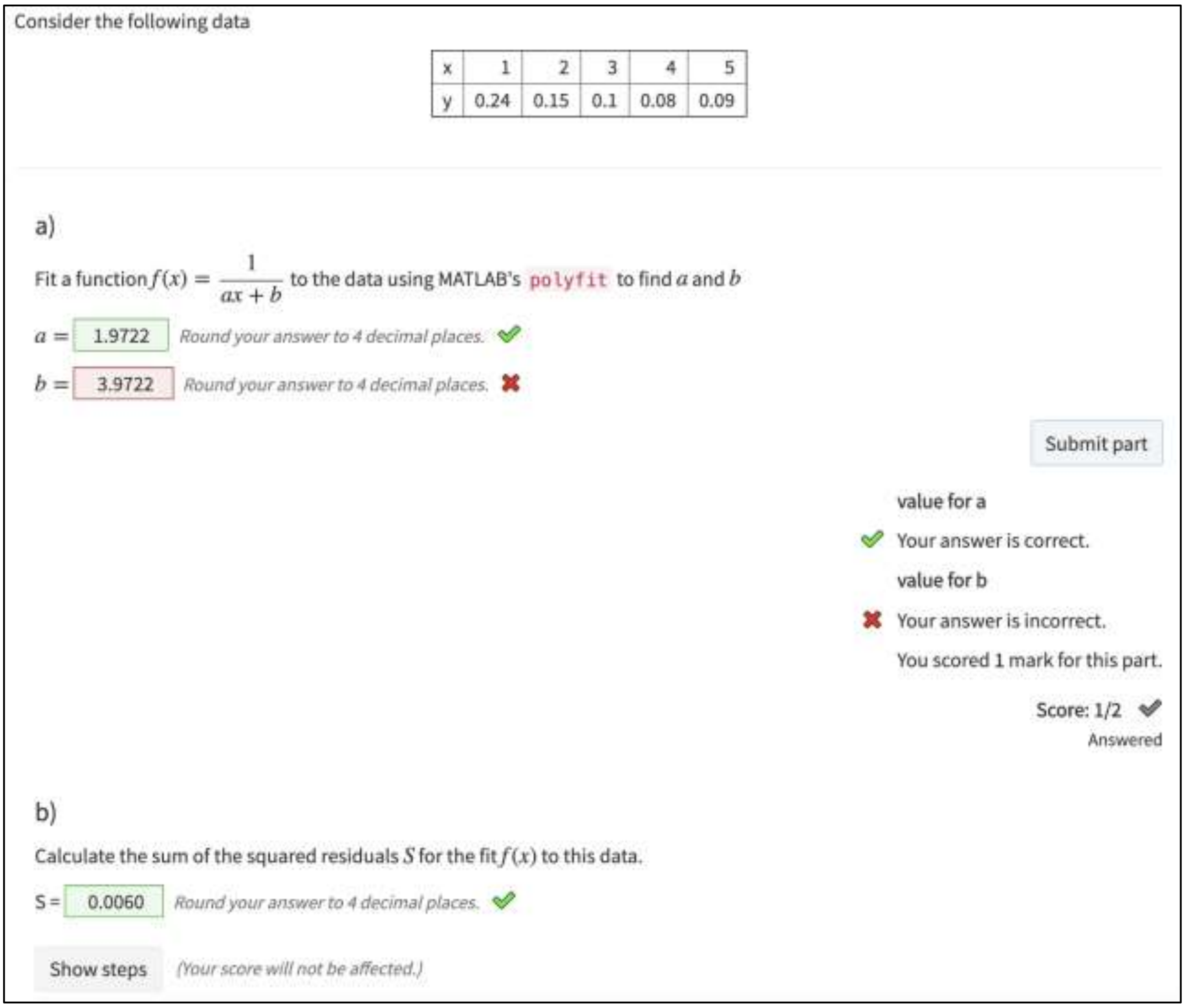
A Numbas practice test was provided for each of the four topics of the numerical methods section in the 2019/20 academic year. These tests have settings such that students receive immediate marking and feedback, can reveal answers and regenerate questions, whilst starting as many attempts as they like. Just over half of the 160 enrolled students took advantage of the tests.

Student feedback on the practice material was very positive, with 80% indicating that the Numbas questions helped them to better understand the course material. Comments included, "the practice material was not only useful for the tests but also stimulating and engaging, which made me very interested in the module".

4.2. Written Assignment

A short assignment forms part of the summative assessment on root finding and curve fitting, for which randomised questions offer some mitigation to the potential problem of solutions being shared. The assignment takes a hybrid approach, randomising each question and automatically marking some of the key results, whilst students are also asked to submit their code and a plot of their results, giving the opportunity to assess the students' coding style and their presentation skills.

Adaptive marking applied to a curve-fitting question


Consider the following data

x	1	2	3	4	5
y	0.24	0.15	0.1	0.08	0.09

a)

Fit a function $f(x) = \frac{1}{ax+b}$ to the data using MATLAB's `polyfit` to find a and b

$a =$ 1.9722 *Round your answer to 4 decimal places.* ✓

$b =$ 3.9722 *Round your answer to 4 decimal places.* ✗

value for a

✓ Your answer is correct.

value for b

✗ Your answer is incorrect.

You scored 1 mark for this part.

Score: 1/2 ✓

Answered

b)

Calculate the sum of the squared residuals S for the fit $f(x)$ to this data.

S = 0.0060 *Round your answer to 4 decimal places.* ✓

(Your score will not be affected.)

Figure 6. A MATLAB curve-fitting question. The student is provided with some random data and asked to fit a curve using MATLAB's *polyfit* function. In this case they need to transform the function into a polynomial with a change of variables before using *polyfit*, so a common error is to fit $f(x) = ax + b$ to the data. With this in mind, part b) uses adaptive marking to award the marks to a student who has gone wrong in part a), but subsequently carried out their calculation of the residuals correctly. Part b) also utilises the steps feature to provide advice on how to make the calculation.

4.3. Class test

The main component of assessment for the numerical methods section of the module is a class test. The class test is open book and students are allowed to use any resources with the exception of communication, through any medium. Like the written assignment, a hybrid test uses some subtle randomisation of Numbas questions, whilst students are also asked to upload their code through the virtual learning environment. Tests like these are held in computer clusters, which are not designed for administrating examinations and do not offer protection from students reading the screen of their neighbour. Though the test is invigilated, the randomisation again offers some additional protection.

Students are provided with a mock exam in the same format. Because some of the marking is automatic, this gives an opportunity to receive instant feedback on their work. All 160 students who sat the class test in the 2019/20 academic year had attempted the mock, with 343 attempts in total, at an average of 63 minutes spent on each attempt. Fifty-eight percent of 81 surveyed students responded that they prefer the online format to a paper mock test, though many commented that, as well as solutions given inside the advice sections of the Numbas questions, they would like the lecturer to work through the mock test in a lecture.

5. Conclusions

The development of Numbas material for computing modules in the mathematics curriculum has been a success to date. Students particularly appreciate the immediate feedback offered by the formative material that is associated with each module topic. However, there is relatively low uptake of the Numbas material compared to the number of student's working through the handouts. At present, practice tests are separated from the handout on the same topic; Numbas can be embedded directly into web pages, and one possible solution is to replace the static questions in handouts with embedded Numbas questions, so that students can get feedback without disrupting their progress through the handout.

The assessment material currently does not extend Numbas to marking computer code directly. This is a limitation, as a hybrid approach is required to directly assess students' coding techniques and, for example, the presentation of results. Students would benefit from feedback on their code, especially in the introductory material for each programming language. Some early attempts have been made to add this functionality by connecting to an instance of the INGInious platform from Numbas, and indeed formative questions making use of this method were popular with students.

Resources

Example questions can be found in the Numbas editor in the *Newcastle University Computing for Mathematics* project (Graham, 2020).

6. Acknowledgments

Thank you to Christian Lawson-Perfect, George Stagg and Lee Fawcett for their help with the design and development of questions.

7. References

Beevers, C.E., Cherry, B.S.G., Clark, D.E.R., Foster, M.G., McGuire, G.R. and Renshaw, J.H., 1988. The CALM before the storm! CAL in university mathematics*. Computers & Education*, 12(1), pp.43-47.

Derval, G., Gego, A., Reinbold, P., Frantzen, B. and Van Roy, P., 2015. Automatic grading of programming exercises in a MOOC using the INGInious platform*. European Stakeholder Summit on experiences and best practices in and around MOOCs (EMOOCS'15).* pp.86-91.

Foster, B., Perfect, C. and Youd, A., 2012. A completely client-side approach to e-assessment and e-learning of mathematics and statistics. *International Journal of e-Assessment*, *2*(2).

Graham, C., 2020. *Newcastle University Computing for Mathematics Numbas Project.* Available at: https://numbas.mathcentre.ac.uk/project/5632/ [Accessed 29 February 2020].

Gwynllyw, R. and Henderson, K., 2009. DEWIS-a computer aided assessment system for mathematics and statistics*. CETL-MSOR Conference 2008*.

Lobb, R. and Harlow, J., 2016. Coderunner: A tool for assessing computer programming skills. *ACM Inroads*, *7*(1), pp.47-51.

Newcastle University, 2020. *Numbas Editor.* Available at: https://numbas.mathcentre.ac.uk [Accessed 25 February 2020].

Sangwin, C., 2015. Computer aided assessment of mathematics using STACK. In: *Selected regular lectures from the 12th International Congress on Mathematical Education*. Springer, Cham. pp. 695-713.

Sangwin, C.J. and O'Toole, C., 2017. Computer programming in the UK mathematics curriculum. *International Journal of Mathematical Education in Science and Technology*, 48(8), pp.1133-1152. https://doi.org/10.1080/0020739X.2017.1315186.

Stagg, G., 2019 *ncldata: NCL example data for use with MAS modules*. Available at: https://rdrr.io/rforge/ncldata/ [Accessed 25 February 2020].

Weir, I., Gwynllyw, R. and Henderson, K., 2017. Creating statistics e-assessments using Dewis with embedded R code*. MSOR Connections*, 15(2), pp.51-59. https://doi.org/10.21100/msor.v15i2.422.

CASE STUDY

Developing computational mathematics provision in undergraduate mathematics degrees

Richard Elwes, School of Mathematics, University of Leeds, Leeds, UK.
Email: r.h.elwes@leeds.ac.uk. https://orcid.org/0000-0002-6752-5501.
Rob Sturman, School of Mathematics, University of Leeds, Leeds, UK.
Email: r.sturman@leeds.ac.uk. https://orcid.org/0000-0001-7299-9931.

Abstract

Over the last ten years we have comprehensively embedded computational mathematics, and in doing so programming, into the undergraduate mathematics degree programmes at the University of Leeds. This case study discusses some of the practical, organisational and pedagogical issues we encountered, and how we addressed them.

Keywords: computational mathematics, Python, employability.

1. Introduction

Although this special issue is about “programming in the mathematics curriculum”, this article is about the more specific topic of computational mathematics. Any definition of “computational mathematics” (beyond “the study and practice of solving mathematical problems with a computer”) might involve terms such as numerical analysis, algorithmic thinking, symbolic manipulation, computer-assisted proof or complexity theory. Certainly each of these topics contains sufficient mathematics, and computation, to fill a course for undergraduates. They might, or might not, also require a student to learn to program a computer.

A review of computer programming provision (Sangwin & O’Toole, 2017) within UK mathematics programmes states that computing is “*more popular amongst applied mathematicians and statisticians*”, with “*very few explicit pure mathematics courses*” within their sample of programming modules. Our view is that undergraduate computational mathematics should be as broad as possible. We argue to our students that there are two good reasons to turn to a computer when confronting a mathematical problem: when you know exactly what you want to do, and when you don’t. The first of these is typically applied mathematics and numerical methods – we have a set of routine calculations to perform, and lack only the time to do these ourselves. The second covers more open-ended problems, possibly with no known solution, where we use a computer to seek evidence, visualise possible solutions, reveal mechanisms and aid our understanding.

Our approach is best illustrated with a typical pair of questions:

1. Write a function that performs Euclid’s algorithm on a pair of positive integers $a > b$, returning their greatest common divisor.
2. Investigate the following results of Lamé and Heilbronn:
    a. The number of steps required by Euclid’s algorithm is at most 5 times the number of digits of b.
    b. On average the number of steps required is equal to $\frac{12\ln 2}{\pi^2}\log_{10} a$ for all pairs a, b.
    c. The number of steps is maximised when a and b are consecutive Fibonacci numbers.

Notice that the question requires the students both to convert a familiar algorithmic process into computer code, and to explore for themselves mathematical facts that are likely to be beyond their current mathematical sophistication to prove. We found it relatively easy, and quite liberating, to devise such questions of this nature, including topics such as the Collatz conjecture, elementary number theory (e.g. Goldbach's conjecture and Euler bricks), continued fractions, iterated function systems, sorting algorithms and cryptography.

Thus our purpose in developing a set of modules on computational mathematics at the University of Leeds was: to introduce mathematical problems which are appropriate for solving with a computer; to see why some problems are *not* suitable for solving in this way; to present some fundamental techniques in mathematical computation; to encourage an investigative spirit in attacking problems which are not intended to be solved completely. We promote this independent approach to learning by scheduling only one hour of lecture time per week for material delivery, leaving the bulk of student time to smaller workshop sessions staffed by a team of demonstrators.

"doing my own research made me feel truly more independent and free to let my mathematical creativity flow, which can't always be the case" (Anonymous student feedback, 2019)

Clearly, studying computational mathematics in this sense inevitably involves learning rudimentary programming techniques. Thus, computational mathematics provides students with an opportunity to acquire valuable transferrable skills, much as delivering oral presentations on mathematical topics provides offers both narrow curricular and broader life-skill benefits. Both activities provide mechanisms to broaden students' range of experience without diluting core mathematical content. Indeed our syllabus often supports the rigorous mathematical content of other modules.

"*I enjoyed the freedom to learn many things through one, gaining a deeper understanding of mathematics, increasing my ability to solve problems, and learning how to program"* (Anonymous student feedback, 2018)

In Leeds, and likely in many places, our introduction of this module was not the very first step towards introducing computation into the mathematics curriculum. There were already a small number of computational components in the form of worksheets using proprietary mathematical software such as Matlab, Maple, and Mathematica. But these were not the best use of limited resources (financial or time) and mutually incompatible. Thus, introducing computational mathematics was not wholly new, but rather the opportunity to replace legacy teaching methods with expanded modern ones.

This case study describes challenges and (our) solutions in devising a practical computational mathematics module (initially), and then a suite of modules. In the spirit of programming, we describe these challenges as bugs, and give our fixes to these problems in the context of a large UK university mathematics department. Of course, such problems may differ in their reproducibility across different institutions, as may the viability of solutions.

2. Planning the module

Bug summary: Colleagues hate the idea.

Bug description: There is distrust among academic colleagues in mathematics that programming is something we *should* introduce into our curriculum. Don't many students choose a degree in Mathematics precisely because they *don't* want to work with computers? Isn't this diverting student (and staff) time away from our core task of learning (and teaching) mathematics?

Fix: There were four arguments we made to colleagues, and the combination was compelling.

(i) Our external advisory board (which comprises local and national employers in a range of industries and services) had given a strong steer that they would value greater computational skills in mathematics graduates. See CBI/Pearson (2019) for development of this point.

(ii) In an environment in which we were encouraged, and keen, to develop our provision of "employability" skills within the mathematics curriculum, programming expertise seemed the natural way to achieve this whilst retaining mathematical content.

(iii) With growing student numbers, and final year projects becoming more individual and resource-intensive, having a student cohort who were confident at investigation was clearly desirable. We argued that computational mathematics formed the ideal place to introduce problems which may not be soluble by an undergraduate mathematician, but which are amenable to experimentation and investigation.

(iv) Importantly, we appealed to a wide cross-section of academic colleagues. This module was never intended to simply be numerical analysis. Both pure and applied mathematicians became convinced that the content of the computational mathematics module was relevant to their field. Elementary number theory provides a wonderful pool of problems for investigation, while automatic theorem proving, an area of pure mathematics in which Leeds has research expertise, has been developed within our undergraduate programming syllabus.

Bug summary: We don't know what language to choose.

Bug description: Institutions may have reasons to choose different languages (the need to prepare students to use particular software in later years; existing proprietorial software deals; the expertise of whoever is developing and teaching the module).

Fix: Several factors led us to choose Python as the single language for the module. Firstly, we wished to move away from proprietary mathematical software. Sangwin & O'Toole (2017) found Matlab to be the most popular language in compulsory year 1 modules mathematics degrees in the UK (of those institutions who responded). Maple is also well represented, and had previously been in use at Leeds within first year modules, albeit in a somewhat peripheral capacity. A comparison between Python and Matlab (and R) in a pedagogical setting is (Ozgur et al., 2017). However, and regardless of any financial incentive, we saw a real opportunity in switching to a general purpose programming language, in that it would allow us to offer our students a genuine transferrable skill, valued by employers (we will return to this point below).

"I feel I have learnt a lot of practical skills to take into the workplace"
(Anonymous student feedback, 2019)

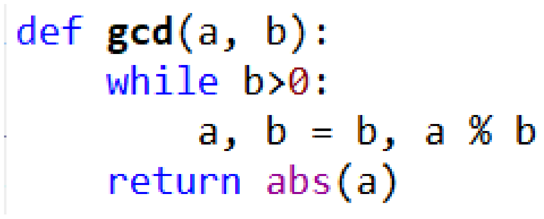
As well as being free and general-purpose, we wanted something with a quick entry time for first-time programmers. Python proved well-suited to this, in our case via the open-source distribution Anaconda and its interface Spyder. We found this straightforward to install on institution-wide cluster machines, and quick and easy for individual students on their own Windows, Linux and Mac OS X machines. Crucially all such installations looked and operated in essentially identical ways.

Python is generally recognised as being a good choice for first-time programmers, a judgement with which we agree: it is dynamically-typed, "duck-typed" (meaning that objects generally behave in accordance with naïve expectation), it has a relatively small vocabulary of keywords most of which are simple English terms, the use of indentation aids readability, it compiles on-the-fly. Thus the journey to the execution of a first "hello world" programme is as quick and simple as for any language (and much more so than, say, C or Java).

A relevant example of Python's duck-typing can be seen by entering "3/2" (dividing the integer 3 by the integer 2). Python 3 automatically outputs a float (1.5). It is easy to take this silent type-change for granted, and in a mathematical setting we often find it convenient to do so (in other cases we deploy integer division 3//2). However, it is also worth spending a moment on the powerful pedagogical point here, that a computer is only ever representing the idea of mathematical quantity, and it may usefully do so in different ways (as illustrated by Python 2 interpreting "/" differently).

Thus we have found that Python represents a better balance than say C or Java, in terms of the relative prominence of syntactic versus mathematical or algorithmic concerns for beginner programmers. This is also illustrated by the kinds of error that such students make. All programming languages are liable both to syntax errors (such as, in Python, confusing "=" and "==") and logic/semantic errors in which (roughly speaking) the code runs or compiles but does not behave as intended. However, our experience is that the relative prevalence of these types of error differs: in C & Java syntactic errors predominate, while in Python beginning students spend longer grappling with semantic/logic errors. We view this as positive, as these errors have greater mathematical/algorithmic educational value.

Python, we argue, allows mathematical and algorithmic aspects to come to the fore rather than being concealed behind technicalities. Consider Figure 1, a Python programme for implementing Euclid's algorithm (and thus a possible solution to question 1 in Section 1). The coding can be done in four short lines using a small number of simple commands, allowing the student to focus on the (significantly more challenging) mathematical matter of understanding how this programme achieves its goal:



```
def gcd(a, b):
    while b>0:
        a, b = b, a % b
    return abs(a)
```

Figure 3: A simple Python function to compute the greatest common divisor of integers *a* and *b*.

Note the single line in which *a* and *b* are re-assigned to the values of *b* and (*a* mod *b*) respectively. Single-line multiple assignment is possible in Python, but not in C or Java, where either the introduction of a temporary variable, or some other method, would be required. Features of this sort, such as list comprehensions, dictionaries and function decorators are typical in Python, and we have found them highly appropriate for computational mathematics. Notice, for instance, the code `Y = [myfunction(x) for x in X]` which illustrates list comprehension is highly reminiscent of applying a mathematical function to every object in a domain `X`, to obtain its range `Y`. Thus `Y = [x**2 for x in range(10)]` is, we claim, not only more elegant than writing a loop, but is more mathematically intuitive. The "Zen of Python" (Peters, 2004) contains several guiding principles which we believe also encapsulate mathematical attitudes: beautiful is better than ugly; explicit is better than implicit*; simple is better than complex; complex is better than complicated.

* We are grateful to the anonymous reviewer for fairly objecting that the earlier remarks about dynamical typing could be seen to contradict this proverb. We might respond by drawing distinctions between explicitness at the level of code (which

Python also carries the benefit of being increasingly popular in the industries in which our graduates typically seek employment. The TIOBE Company maintains the *TIOBE Programming Community Index*, a popularity metric for programming languages as measured by a variety of internet search tools. Figure 2 puts Python firmly in third place, behind Java and C, but growing significantly in popularity in recent years.

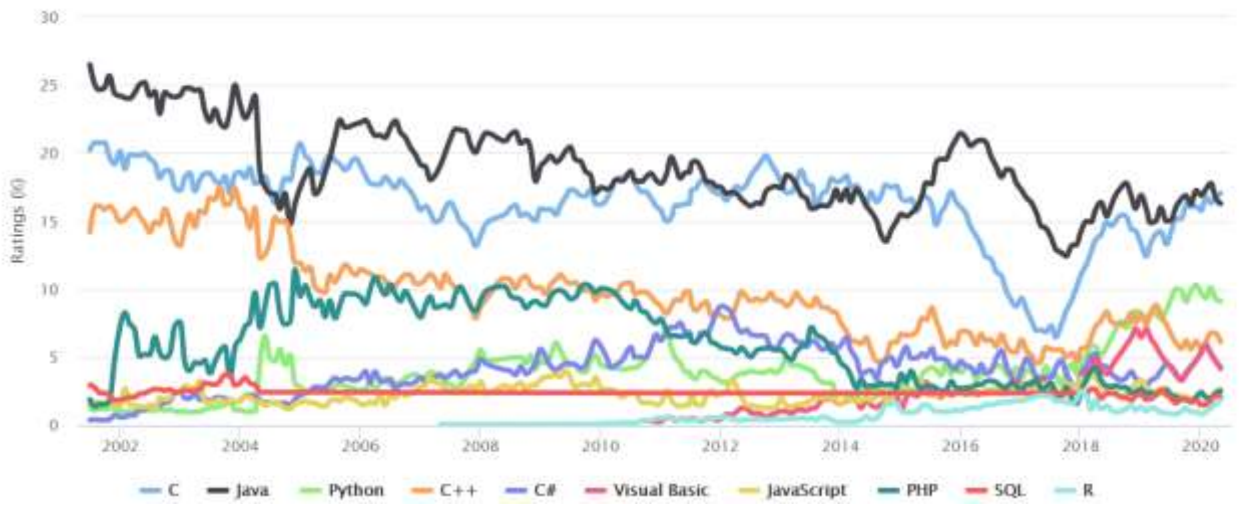


Figure 4: Popularity ratings of various languages from the TIOBE Programming Community (TIOBE, 2020)

Bug summary: I worry about accessibility and inclusivity.

Bug description: The particular software used may not be fully accessible and care must be taken to meet the needs of all students.

Fix: Of course solutions need to be personalised to meet the individual needs of students. We give a particular example of a visually impaired student who was worried about being able to use the software efficiently and effectively. After listening to her describe her particular requirements, we were able to help her set up her interface, using a combination of control key-strokes and magnification. She found Spyder more accessible than notebooks, and said:

"*Firstly, I'd set my screen up so that it was split vertically with the python console on one side and the editor window on the other side. I found this was the best way to fit the most information on the screen when you'd enlarged the text. To enlarge the text I would [use key-strokes] to zoom in to specific sections of the code, [together with] the standard Windows magnifier [to navigate the software features]. Using these two magnification techniques I managed to navigate my way around Spyder and the specific code quite effectively*".

---

is desirable) versus at the level of the language itself (where the suppression of declaration of variables is convenient), and between explicitness in describing procedures versus that describing objects. However, we do not have the space to develop such arguments further.

We also note that there are communities working towards the inclusivity and accessibility of Project Jupyter's software, including iPython notebooks (GitHub, 2019).

3. Running the module

Bug summary: Marking students' work is time-consuming

Bug description: With a class of over 300 students, each submitting 10 pieces of coursework (3 summative, 7 formative) through the semester, our module undeniably entailed a great deal of marking.

Fix: Recruiting extra staff to help with the marking is likely an unavoidable part of the solution (our module has a team of 5 or 6 markers, either postgraduate students or tutorial assistants). Nevertheless, the use of appropriate technology has allowed us to bring the problem down to manageable proportions (and we hope for further improvements to come).

In 2019, we worked with colleagues Craig Evans and Samuel Wilson from the School of Electronic and Electrical Engineering, to adapt their automatic feedback and assessment platform (Evans and Wilson 2019) to the specific needs of our module. This system tests the Python functions within submissions against specified inputs, generating a feedback text file for each student. It is thus automatic to judge functions as being correct, and to identify common errors and award partial marks accordingly. Human input is concentrated in places where it is essential: in setting up the marking system, in identifying uncommon errors (and hopefully thereby improving the system), in making sense of error-strewn submissions, and most valuably in assessing styles of questions which cannot be done automatically. These include plotting graphs, commenting on the results of experiments, performing open-ended investigations such as question 2 in section 1. (We remark that solutions to such questions are also much harder to plagiarise, which forms an important part of our strategy in that regard.)

Another benefit of automatic marking has been that students get accustomed to working under tighter instructions: in our system functions must be given specific names, they must accept inputs of a given type, and they must output their results in a specified format. Any deviation from these requirements will cause the automatic marker to fail, and students are now used to marks being lost in such cases.

4. Developing computational mathematics further

Bug summary: After the module, students want to do more computational mathematics.

Bug description: This should be welcomed as a success, of course. Nevertheless, there is a risk of either disappointing enthusiastic students or creating additional pressure on already crowded timetables.

Fix: Although it would be possible to create an entire follow-on module entitled “Advanced Computational Mathematics”, there was little appetite for this among staff. Rather our approach has been to embed further computation in pre-existing modules, in two ways.

The computational mathematics module we have discussed runs in the first semester of second year. We created computational ‘add-ons’ to existing second year (second semester) modules. There are currently three such, that have been brought in at different times. In each case, a pre-existing 10 credit module was supplemented with a new 5 credit computational part. Students can either study the original 10 credit theoretical module or the new 15 credit “with computation” variant.

We have such modules in Numerical Analysis, Discrete Mathematics, and Logic. In each there are twin aims: to apply computational methods in the given setting, and to develop Python techniques for which there was little or no time in the Computational Mathematics module. For instance, in Discrete Mathematics with Computation, students learn to build recursively defined functions; in Numerical Analysis with Computation students construct new classes for rational numbers and learn how to handle vectors and matrices efficiently.

All Leeds students undertake a research project which runs through both semesters of their final year. Third year mathematicians select their project from a published list of over 50 titles. These vary widely in subject-matter (pure, applied, statistical, financial) and in the tools and techniques required. However, several such projects require computation of assorted kinds, in either major or minor ways. Indeed, one of the goals of the introduction of our Computational Mathematics module was to allow the scope of such projects to be broadened. For example, there are projects on discrete random processes, automatic puzzle-solving, and numerical equation solving. Students' prior knowledge of Python has undoubtedly extended the possibilities here. To broaden the range still further and cater to the most enthusiastic students, we initiated a dedicated project-title on “Computational Applied Mathematics” in which students select from several topics including implementing Fast Fourier Transforms and modelling a nonlinear pendulum.

5. References

CBI/Pearson, 2019. *Education and learning for the modern world*. Available at: https://www.cbi.org.uk/media/3841/12546_tess_2019.pdf [Accessed 29 May 2020].

GitHub, 2019. *jupyter / accessibility*. Available at: https://github.com/jupyter/accessibility [Accessed 29 May 2020].

Ozgur, C., Colliau, T., Rogers, G., Hughes, Z. and Myer-Tyson, B., 2017. MatLab vs. Python vs. R. *Journal of Data Science*, *15*(3), pp.355-372. Available at: http://www.jds-online.com/volume-15-number-3-july-2017 [Accessed 29 May 2020].

Evans, C.A., and Wilson, S.S., 2019. Development of an automated feedback and assessment platform. *Horizons In Stem Higher Education Conference, Kingston University, 3 July 2019*. Available at: https://ukstemconference.files.wordpress.com/2019/06/horizons-conf-19-abstract-booklet-.pdf [Accessed 29 May 2020].

Sangwin, C.J. and O'Toole, C., 2017. Computer programming in the UK mathematics curriculum. *International Journal of Mathematical Education in Science and Technology*, 48(8), pp.1133-1152. https://doi.org/10.1080/0020739X.2017.1315186.

Peters, T., 2004. *The Zen of Python*. Available at: https://www.python.org/dev/peps/pep-0020/ [Accessed 29 May 2020].

TIOBE, 2020. *TIOBE Programming Community Index*. Available at: https://www.tiobe.com/tiobe-index/ [Accessed 29 May 2020].